НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ННК «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра системного проектування
(повна назва кафедри)

«На правах рукопису»
УДК _004.852

«До захисту допущено»
Завідувач кафедри
_____ _А.І. Петренко
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації)__122 - комп'ютерні науки та інформаційні технології «Системне проектування сервісів»_____
(код і назва спеціальності)

на тему: Реєстр веб-сервісів з семантичними анотаціями для мобільної медичної платформи

Виконав: студент _6_ курсу, групи ДА-62м
(шифр групи)

_____Івченко Дмитро Анатолійович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник ____зав. каф., професор, д.т.н. Петренко А.І_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Розроблення стартап-проекту

_____зав. каф., професор, д.т.н. Петренко А. І._____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Інститут/факультет Інститут прикладного системного аналізу
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 122 - комп'ютерні науки та інформаційні технології «Системне проектування сервісів»
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І. Петренко
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Івченку Дмитру Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації « Реєстр веб-сервісів з семантичними анотаціями для мобільної медичної платформи»
науковий керівник дисертації Петренко А.І., проф.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження Семантично орієнтовані веб-сервіси

4. Предмет дослідження Реєстр веб-сервісів з семантичними анотаціями для мобільної медичної платформи

5. Перелік завдань, які потрібно розробити провести огляд існуючих існуючих технологій для створення семантичних веб-сервісів, розробити веб сервіс, що

буде відповідати вимогам, оформити роботу на основі отриманих результатів

6. Орієнтовний перелік публікацій Івченко Д.А. Використання Neo4j і GraphQL на базі GraphDB для створення мікросервісу / Д. А. Івченко. // System Analysis and Information Technologies – 2018. – С. 171–174

7. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Реалізація стартап-проекту			

8. Дата видачі завдання 01.02.2018

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.02.2018	
2	Збір інформації та аналіз літератури	15.02.2018	
3	Проведення огляду існуючих методів побудови семантичних веб-сервісів	28.02.2018	
4	Формалізація задачі розробки семантичного веб-сервісу	11.03.2018	
5	Реалізація поставленої задачі	13.04.2018	
6	Аналіз та порівняння результатів моделювання	25.04.2018	
7	Оформлення дипломної роботи	30.04.2018	
8	Отримання допуску до захисту та подача роботи в ДЕК	09.05.2018	

Студент _____ Івченко Д.А.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Петренко А.І.
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

РЕФЕРАТ НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ

виконану на тему: Реєстр веб-сервісів з семантичними анотаціями для мобільної медичної платформи

студентом: Івченко Дмитром Анатолійовичем

Робота виконана на 95 сторінках, містить 42 ілюстрацій, 22 таблиць. При підготовці використовувалась література з 28 джерел.

Актуальність теми

Розробки систем у сфері медицини останнім часом набула високої популярності, тому наразі є дуже важливим робити апаратне забезпечення таким або воно було конкуренто спроможним на ринку. Мікро сервіси на базі графових баз даних уже давно мають місце в таких компаніях як Amazon, IBM чи Microsoft.

Мета та задачі дослідження

Провести огляд існуючих існуючих технологій для створення семантичних веб-сервісів, розробити веб сервіс для зберігання даних про результати аналізів пацієнтів клініки, що буде відповідати вимогам, оформити роботу на основі отриманих результатів.

Рішення поставлених завдань та досягнуті результати

У даній роботі було представлено інноваційний метод створення веб-сервісів для мобільних додатків на базі використання Neo4j і GraphQL , що обслуговує графову базу даних GraphDB. Кожен вузол (суб'єкт або атрибут) в моделі графових баз даних безпосередньо і фізично містить список взаємопов'язаних записів, які представляють його зв'язок з іншими вузлами. Всякий раз, коли запускається еквівалент операції JOIN, база даних просто використовує цей список і має безпосередній доступ до пов'язаних вузлів, що усуває необхідність в дорогому обчисленні пошуку / зіставлення. Розроблено

сайт для наповнення даної бази знань на базі Grand Stack (React, Neo4j, Apollo, GraphQL).

Об’єкт досліджень

Семантичні веб сервіси

Предмет досліджень

Методи та варіанти створення семантичних веб сервісів.

Методи досліджень

Для вирішення проблеми в даній роботі використовуються методи аналізу і синтезу, системного аналізу, порівняння, логічного узагальнення результатів.

Наукова новизна

Наукова новизна роботи полягає у використанні інноваційного підходу до створення семантичного веб сервісу на базі Neo4j, GraphQL та GraphDB.

Практичне значення одержаних результатів

Отриманий веб сервіс може бути використано як частину великої екосистеми мікро сервісів. Також сам приклад створення є достатньо детально описаним, щоб створювати інші мікро сервіси на базі нього.

Публікації

Івченко Д.А. Використання Neo4j і GraphQL на базі GraphDB для створення мікро сервісу / Д. А. Івченко. // System Analysis and Information Technologies. – 2018. – С. 171–174

Ключові слова

Семантичний веб сервіс, мікро сервіс, Neo4j, GraphQL, GraphDB, mutations, query.

Abstract on master's thesis

on topic: The web services catalogue with semantic annotations for mobile health platform

student: Dmytro A Ivchenko

Work carried out on 95 pages containing 42 figures, 22 tables. The paper was written with references to 28 different sources.

Topicality

The development of systems in the field of medicine has recently become very popular, so it is now very important to make hardware such or that it was competitive in the market. Micro services based on graph databases have long taken place in companies such as Amazon, IBM or Microsoft.

Purpose

Conduct a review of existing existing technologies for creating semantic web services, develop a web service for storing data on the results of patients' clinical examinations that will meet the requirements, formulate work based on the results obtained.

Solution

In this paper, was presented an innovative method for creating Web services for mobile applications based on the use of the Neo4j and GraphQL, which serves the GraphDB graph database. Each node (subject or attribute) in the graph database model directly and physically contains a list of interconnected records that represent its connection with other nodes. Whenever the equivalent JOIN operation is triggered, the database simply uses this list and has direct access to related nodes, eliminating the need for expensive search / comparison computing. A site for filling this knowledge base on the basis of Grand Stack (React, Neo4j, Apollo, GraphQL) has been developed.

Object of research

Semantic web services.

Subject of research

Methods and options for creating semantic web services.

Research methods

To solve the problem in this paper we use methods of analysis and synthesis, system analysis, comparison, logical generalization of the results.

Scientific novelty

The scientific novelty of the work is to use an innovative approach to creating a semantic Web service based on Neo4j, GraphQL and GraphDB.

The practical value of the results

The resulting web service can be used as part of a large micro-services ecosystem. Also, the creation example itself is described in detail in order to create other micro services based on it.

Publications

Ivchenko D. Neo4j i GraphQL on the top of GraphDB for creationg of microservice. / D. Ivchenko. // System Analysis and Information Technologies. - 2018. - P. 171-174

Keywords

Semantic web service, microservice, Neo4j, GraphQL, GraphDB, mutations, query

РЕФЕРАТ НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

выполненную на тему: Регистрация веб-сервисов с семантическими аннотациями для мобильной медицинской платформы

студентом: Ивченко Дмитрием Анатольевичем

Работа выполнена на 95 страницах, содержит 42 иллюстраций, 22 таблиц. При подготовке использовалась литература с 28 источников.

Актуальность темы

Разработки систем в области медицины в последнее время приобрела высокую популярность, поэтому сейчас очень важно делать аппаратное обеспечение таким или оно было конкурентоспособности способным на рынке. Микро сервисы на базе графовых баз данных уже давно имеют место в таких компаниях как Amazon, IBM или Microsoft.

Цель и задачи исследования

Провести обзор существующих существующих технологий для создания семантических веб-сервисов, разработать веб сервис для хранения данных о результатах анализов пациентов клиники, будет соответствовать требованиям, оформить работу на основе полученных результатов.

Решение поставленных задач и достигнутых результатах

В данной работе был представлен инновационный метод создания веб-сервисов для мобильных приложений на базе использования Neo4j і GraphQL, обслуживающего графовую базу данных GraphDB. Каждый узел (субъект или атрибут) в модели графов баз данных непосредственно и физически содержит список взаимосвязанных записей, представляющих его связь с другими узлами. Всякий раз, когда запускается эквивалент операции JOIN, база данных просто использует этот список и имеет прямой доступ к связанным узлов, что устраняет необходимость в дорогом исчислении поиска / сопоставления. Разработан сайт

для наполнения данной базы знаний на базе Grand Stack (React, Neo4j, Apollo, GraphQL).

Объект исследований

Семантические веб сервисы

Предмет исследований

Методы и варианты создания семантических веб сервисов.

Методы исследований

Для решения проблемы в данной работе используются методы анализа и синтеза, системного анализа, сравнения, логического обобщения результатов.

Научная новизна

Научная новизна работы заключается в использовании инновационного подхода к созданию семантического веб сервиса на базе Neo4j, GraphQL и GraphDB.

Практическое значение полученных результатов

Полученный веб сервис может быть использовано как часть большой экосистемы микро сервисов. Также сам пример создания достаточно подробно описанным, чтобы создавать другие микро сервисы на базе него.

Публикации

Ивченко Д.А. Использование Neo4j і GraphQL на базе GraphDB для создания микро сервиса / Д. А. Ивченко. // System Analysis and Information Technologies. - 2018. - С. 171-174

Ключевые слова

Семантический веб сервис, микро сервис, Neo4j, GraphQL, GraphDB, mutations, query.

ЗМІСТ

Список умовних позначень, символів, скорочень та термінів

SWS – Semantic Web Service
OWL-S – Semantic Markup for Web Services
WSML – Web Service Modeling Language
RDF – Resource Description Framework

1 Опис семантичного веб-сервісу

1.1 Вступ

Технологія семантичного веб-сервісу (SWS) виявляє конвергенцію (збіжність) семантичного Інтернету із сервіс-орієнтованими обчисленнями. Вона вирішує головну проблему автоматизованої, між операційної та змістовної координації веб-сервісів, які виконуються інтелектуальними програмними агентами. У цьому розділі коротко розглянемо відомі структури описання SWS, а саме такі стандартні мови як: SAWSDL (семантичне анотування для мови описання веб-сервісів), OWL-S (мова описання онтології для семантичного сервісу) та WSML (мова моделювання веб-сервісів).

1.2 Проблеми опису семантичного сервісу

Кожну структуру опису семантичного сервісу можна охарактеризувати з урахуванням того, (а) яка семантика сервісу описується, (б) якою мовою або формалізмом, та (в) які обґрунтування щодо описання абстрактних сервісів беруться до уваги. Крім того, ми розрізняємо абстрактний веб-сервіс, тобто описання обчислювальної одиниці сервісу та конкретний сервіс як один з його екземплярів або викликів, які дають реальну користь користувачеві [21]. У цьому значенні, опис абстрактних послуг вважається повним, але не обов'язково правильним: можуть бути конкретні екземпляри сервісу, які є моделями описання можливостей абстрактного сервісу, але фактично провайдери не можуть їх надавати.

1.2.1 Функціональна та нефункціональна семантика сервісу

Загалом, функціональність сервісу може бути описана з точки зору того, що вона робить і як це насправді працює. Обидва аспекти функціональної

семантики (або можливості) збираються в профілі сервісу, відповідно, сервісної моделі процесу. Семантика описує сервіс з точки зору його вхідного (I) та вихідного (O) параметрів та його передумови (P) та ефекти (E), які можуть мати місце до або після реалізації сервісу в певній предметній області, а також деякі додаткові відомості про походження, такі як назва сервісу, його бізнес-домен та хто є провайдером. Модель процесу атомарних або комплексних сервісів описує, як працює сервіс з точки зору взаємодії між даними та потоком управління на основі спільного набору конструкцій робочого потоку або управління, такі як послідовність, методи split + join, вибір та інші.

Ця загальна відмінність між семантикою профілю та моделлю процесу притаманна структурованим моделям опису веб-сервісу, тоді як розбіжності є в найменуванні та формальному представленні того, що є частиною семантики сервісу. Ми можемо надалі розрізняти описання абстрактних сервісів без зберігання стану (IO), та, відповідно, зі зберіганням стану (PE), що представляють собою набір екземплярів, які є конкретними сервісами, корисними для користувачів. Нефункціональна семантика сервісу, як правило, описана з урахуванням якості сервісів (QoS), включаючи обмеження доставки, модель вартості з правилами ціноутворення, відмови, доступності та політики конфіденційності.

1.2.2 Структуроване представлення семантики сервісу

Незалежне від домену та структуроване подання семантики сервісу пропонується сервісними онтологіями та мовами верхнього рівня, такими як OWL-S та WSML з формальними логічними прив’язками, або SAWSDL, по суті, без будь-якої формальної семантики. Ні OWL-S, ні WSML не надають ніякої узгодженої формальної семантики, але інтуїтивно зрозумілу, стандартну семантику на основі робочого потоку сервісної моделі процесу («оркестровка та хореографія»). Як альтернатива, для опису абстрактних сервісів, заснованих на

WSDL, модель процесу може інтуїтивно відображатися на «оркестровці» мові виконання бізнес-процесів BPEL з певною формальною семантикою.

1.2.3 Монолітне представлення семантики сервісу

Формальної специфікації семантики сервісу, що не залежить від будь-якого формату описування структурованого сервісу, можна досягти, наприклад, за допомогою певного набору концептуальних та рольових аксіом у відповідному типу логіки. Оскільки робота сервісу описується за допомогою однієї єдиної концепції сервісу, це представлення семантики сервісу називається монолітним і дозволяє визначити семантичні відносини між повним описом сервісів в рамках логічного формалізму на основі концепції задоволеності, віднесення до певної категорії та слідування. Однак це не надає будь-якої додаткової інформації про те, як сервіс насправді працює з точки зору моделі процесу, а ні будь-якого описання нефункціональної семантики.

1.2.4 Семантика даних

Предметно-залежна семантика параметрів профілю сервісу (також називається семантикою даних) описується в концепціях, ролях (та правилах), взятих з предметної області, завдання, або онтологій додатків. Ці онтології визначаються формальною семантичною веб-мовою, такою як OWL, WSML або SWRL. При використанні різних онтологій, агенти повинні автоматично вирішувати проблему структурної та семантичної неоднорідності для взаємодії, щоб полегшити відкриття та побудову веб-сервісу. Цей процес зіставлення онтологій зазвичай обмежується онтологіями, які вказані на одній і тій же мові, в іншому випадку агентам повинні бути надані відповідні між-онтологічні відображення.

1.2.5 Обґрунтування щодо описання семантичного сервісу

Основна ідея формально-прив'язаних описів веб-сервісу – це дозволити агентам краще зрозуміти функціональну та нефункціональну семантику через відповідні логічні обґрунтування. З цією метою зазвичай передбачається, що застосований тип логічного обґрунтування відповідає основній моделі описання семантичного сервісу. Крім того концепції, що використовуються для визначення семантики даних вхідних і вихідних параметрів сервісу, мають створюватися на базових поняттях та функціях, взятих з формальних онтологій додатків або онтологій предметної області, на які зазвичай посилаються замовники та провайдери. В наступному розділі ми досліджуємо підходи до нелогічного, логічного та гібридного методів обґрунтування для відкриття семантичних веб-сервісів та їх побудови.

1.3 Семантичні анотації для WSDL та XML схеми (SAWSDL)

Стандартна мова WSDL для веб-сервісів працює виключно на синтаксичному рівні, оскільки їй не вистачає будь-якої декларативної семантики, необхідної для цілеспрямованого представлення та обґрунтування за допомогою логічних висновків. Вперше зіткнувшись з цією проблемою, Робоча група Консорціума Всесвітньої павутини (W3C) з питань семантичного анотування для мови WSDL та XML-схеми (SAWSDL) розробила механізми, за допомогою яких можна додавати до компонентів WSDL семантичні анотації (розмітки). Специфікація SAWSDL стала потенційною рекомендацією Консорціуму W3C від 26 січня 2007 року (http: //www.w3.org/2002/ws/sawsd), а зрештою – рекомендацією Консорціуму W3C від 28 серпня 2007 року.

1.3.1 Компоненти розмітки WSDL

На відміну від мови OWL-S або WSML, SAWSDL не є новою мовою або онтологією верхнього рівня для описання семантичного сервісу, а просто забезпечує механізми, за допомогою яких онтологічні поняття, які визначаються

за межами документів WSDL, можуть бути віднесені до семантично анотованих елементів описування WSDL. На ґрунті свого попередника та поданні WSDL-S членами W3C в 2005 році, ключові принципи розробки SAWSDL полягають у тому, що: (а) специфікація уможливлює семантичні розмітки веб-сервісів, які використовуються і будуються на основі існуючої системи розширення WSDL; (б) не залежить від семантичних (онтологічних) мов представлення; і (с) дозволяє семантичні анотації веб-сервісів не тільки для виявлення веб-сервісів, але також для їх виклику.

На основі цих принципів розробки, SAWSDL визначає наступні три нових атрибути розширювання елементів WSDL 2.0 для їх семантичної розмітки:

- Атрибут розширення під назвою *modelReference* для визначення асоціації між компонентом WSDL та концепцією в деякій семантичній (доменній) моделі. Атрибут *modelReference* використовується для розмітки визначень складного типу, простого типу, оголошень елементів та атрибутів XML-схеми, а також інтерфейсів, операцій та несправностей WSDL. Кожен *modelReference* ідентифікує поняття в семантичній моделі, яка описує елемент, до якої він прикріплений.
- Два атрибути розширення (liftingSchemaMapping та downingSchemaMapping) додаються до набору оголошень елементів XML-схеми, визначень складного та простого типу. Обидва дозволяють визначити відображення семантичних даних у домені, на який посилається modelReference і XML, та які можна використовувати під час виклику сервісу.

Приклад сервісу SAWSDL, тобто семантично анотований сервіс WSDL з посиланнями на зовнішні онтології, що описують семантику елементів WSDL, наведено на Рисунку 1.1.

```
<wsdl:types>
 <xs:schema targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#" elementFormDefault="qualified">

    <xs:element name="OrderRequest"
          sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest"
          sawsdl:loweringSchemaMapping="http://www.w3.org/2002/ws/sawsdl/spec/mapping/RDFOnt2Request.xml">
          <xs:complexType>
              <xs:sequence>
              <xs:element name="customerNo" type="xs:integer" />
              <xs:element name="orderItem" type="item" minOccurs="1" maxOccurs="unbounded" />
             </xs:sequence>
         </xs:complexType>
   </xs:element>

<xs:complexType name="item">
     <xs:all> <xs:element name="UPC" type="xs:string" /> </xs:all>  <xs:attribute name="quantity" type="xs:integer" />
</xs:complexType>

<xs:element name="OrderResponse" type="confirmation" />
    <xs:simpleType name="confirmation"
          sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderConfirmation">
         <xs:restriction base="xs:string">
         <xs:enumeration value="Confirmed" />
         <xs:enumeration value="Pending" />
         <xs:enumeration value="Rejected" />
   </xs:restriction>
   </xs:simpleType>
</xs:schema>
</wsdl:types>
```

Рис. 1.1 - Приклад семантичного анотування елементів WSDL в SAWSDL

1.3.2 Обмеження даної анотації

Найбільшій критиці підлягає той факт, що SAWSDL є просто синтаксичним розширенням WSDL, без будь-якої формальної семантики. На відміну від OWL-S та (частково) WSML не існує визначеного формального обґрунтування ані компонентів сервісу WSDL на основі XML, ані посилання на зовнішні джерела метаданих. Цитуючи специфікацію SAWSDL: «Знову ж таки, якщо структури XML, що очікуються клієнтом і сервісом, відрізняються, схематичне зіставлення може перекласти структури XML в семантичну модель, де будь-які невідповідності можна зрозуміти та вирішити». Це робить будь-яку форму логічного виявлення та компоненти сервісу SAWSDL в семантичній павутині досить застарілими, але передбачає «магічних» посередників для вирішення семантичних неоднорідностей.

Ще однією проблемою SAWSDL сьогодні є – за винятком платформи METEOR-S, створеної розробниками SAWSDL (WSDL-S), та постійних витрат ресурсів на розробку в корпорації IBM – все ще дуже обмежена підтримка програмного забезпечення в порівнянні зі значними інвестиціями в дослідження

та розробку програмного забезпечення для більш прогресивних платформ, таких як OWL-S та WSMO по всьому світу. Однак останнє оприлюднення Консорціумом W3C документу щодо SAWSDL в якості рекомендації не тільки в цілому підтримує розвиток платформи веб-сервісу W3C (замість того, щоб перейти на більш прогресивні технології, такі як OWL-S або WSML), але, безумовно, підштовхне розробку програмного забезпечення на підтримку SAWSDL та посилить дослідження щодо перепроектування цих платформ, враховуючи SAWSDL.

1.4 Опис онтології OWL-S

OWL-S – це верхня онтологія, що використовується для описання семантики сервісів на базі стандартної онтології OWL Консорціуму W3C, заснованої на WSDL. Вона має своє коріння в онтології сервісу DAML (DAML-S), випущеного в 2001 році, і стала потенційною рекомендацією Консорціуму W3C в 2005 році. OWL-S будується на вершині OWL і складається з трьох основних онтологій: профіль, модель процесу та обґрунтування (див. Рис. 1.2).

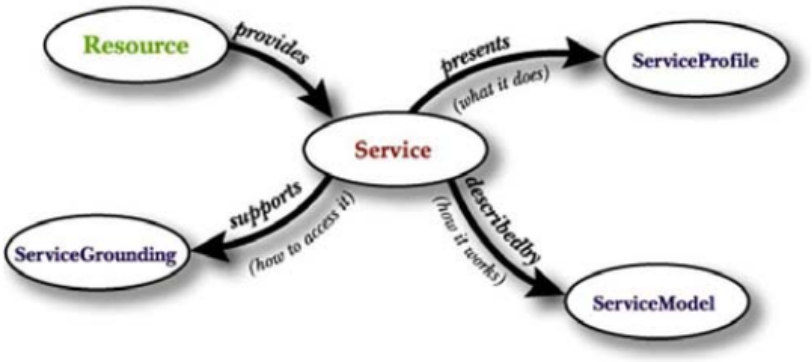


Рис. 1.2 - Елементи описування сервісу OWL-S

1.4.1 Підґрунтя: OWL

Стандартна мова онтології для семантичного веб-сервісу є OWL [2, 4, 12], яка формально ґрунтується на описовій логікі (Description Logic). OWL має свої

корені в спільній ініціативі дослідників DAML + OIL з США та Європи в 2000 році для розробки формальної анотації або мови розмітки для веб-сервісу. Лише через три роки OWL стала рекомендацією Консорціуму W3C, і з того часу була широко прийнята в промисловій та науковій галузях.

1.4.2 Різновиди

OWL має декілька різновидів: OWL-Full, OWL-DL та OWL-Lite. Кожен різновид відповідає DL різної виразності та складності. OWL-Lite та OWL-DL є абстрактною синтаксичною формою дискрипційної логіки SHIF (D), та відповідно, SHOIN (D). Найбільш виразний різновид OWL-Full забезпечує повну сумісність зі схемою RDFS і охоплює виразність діалекту SHOIQ (D), що пропонує не лише прості типи даних (D), а також примітивні транзитивні ролі в потужності множини обмеженого використання (Q) та похідні класи разом з непримітивними ролями (Рисунок 1.3).

Синтаксичне перетворення з онтологій OWL-Lite та OWL-DL на відповідні бази знань DL є поліноміальною складністю.

DL Expressiveness	DL Syntax	DAML/XMLS Syntax	Serv. Descript. Lang.
$\mathcal{ALC}$, also called $\mathcal{S}$ when transitevely closed primitive roles are included	A	daml:Class	Concept
	$\top$	daml:Thing	Thing
	$\bot$	daml:Nothing	Nothing
	$(C \sqsubseteq D)$	daml:subClassOf	Subsumption
	$(C \equiv D)$	daml:sameClassAs	Equivalence
	R	daml:Property	Properties
	R	daml:ObjectProperty	ObjectProperties
	$(C \sqcap D)$	daml:intersectionOf	Conjunction
	$(C \sqcup D)$	daml:disjunctionOf	Disjunction
	$\neg C$	daml:complementOf	Negation
	$\forall R.C$	daml:toClass	Universal Role Rest.
	$\exists R.C$	daml:hasClass	Existential Role Rest.
$\mathcal{N}$	$\leq nR.\top$	daml:maxCardinality	Non-Qualified Card.
	$\geq nR.\top$	daml:minCardinality	
	$=nR.\top$	daml:cardinality	
$\mathcal{Q}$	$\leq nR.C$	daml:hasClassQ daml:minCardinalityQ	Qualified Cardinality
	$\geq nR.C$	daml:hasClassQ daml:maxCardinalityQ	
	$=nR.C$	daml:hasClassQ daml:cardinalityQ	
$\mathcal{I}$	R^-	daml:inverseOf	Inverse Roles
$\mathcal{H}$	$(R \sqsubseteq S)$	daml:subPropertyOf	Subsumption of Roles
	$(R \equiv S)$	daml:samePropertyAs	Equivalence of Roles
$\mathcal{O}$	$\{o\}$	XML Type + rdf:value	Nominals
	$\exists T.\{o\}$	daml:hasValue	Value Restrictions
(D)	D	daml:Datatype + XMLS	Datatype System
	T	daml:datatypeProperty	Datatype Property
	$\exists T.d$	daml:hasClass + XMLS Type	Exist. Datat. Rest.
	$\forall T.d$	daml:toClass + XMLS Type	Univ. Datat. Rest.

Рис. 1.3 - DL-конструктори різновидів OWL (SHIF, SHOIN, SHOIQ)

1.4.3 Відношення до RDFS

Абстрактний синтаксис OWL може бути віднесений до нормативного синтаксису RDF. OWL додає конструктори до RDFS для побудови класів та характеристик властивостей (словника) та нових аксіом (обмежень) з теоретико-модельною семантикою. Тобто OWL розширює виразність RDFS. Зокрема, фрагмент RDFS різновиду OWL-DL не дозволяє, наприклад, стверджувати, що властивість *P* є транзитивною або протилежною до іншої властивості *Q*, і використовуючи перетин (об'єднання) в межах описів (під-)класу, або універсальні / екзистенціальні квантифікації в межах супер- / підкласів [13].

Лише нещодавно було показано [20], що формальна семантика підмови RDFS є сумісною з відповідним фрагментом OWL-DL, таким чином RDFS дійсно може служити фундаментальною мовою семантичного комплексу веб-технологій. Однак перевірити, чи є граф RDF онтологією OWL, та оновити RDFS до OWL на практиці все ще важко. Для детального розгляду цього питання зверніться до додаткових матеріалів [7].

Для OWL-Lite та OWL-DL, виконуваність концепції та цілісність ABox можна розв’язати в класі складності EXPTIME, та відповідно в класі складності NEXPTIME [11, 25]. Хоча SHOIQ (D) з примітивними неперехідними ролями є несумісним з NEXPTIME [25], його різновид з непримітивними транзитивними ролями, а саме OWL-Full, є нерозв’язним [4]. Те саме стосується і підмножини SHN + OWL-DL з обмеженнями потужності множини (N) та ієрархії ролей (H).

Рисунок 1.4 показує співвідношення OWL до інших виділених (поліноміально зменшуваних) розв’язних підмножин DL, таких як EL ++, Horn-SHIQ та DLP разом з відповідними результатами комплексності [2].

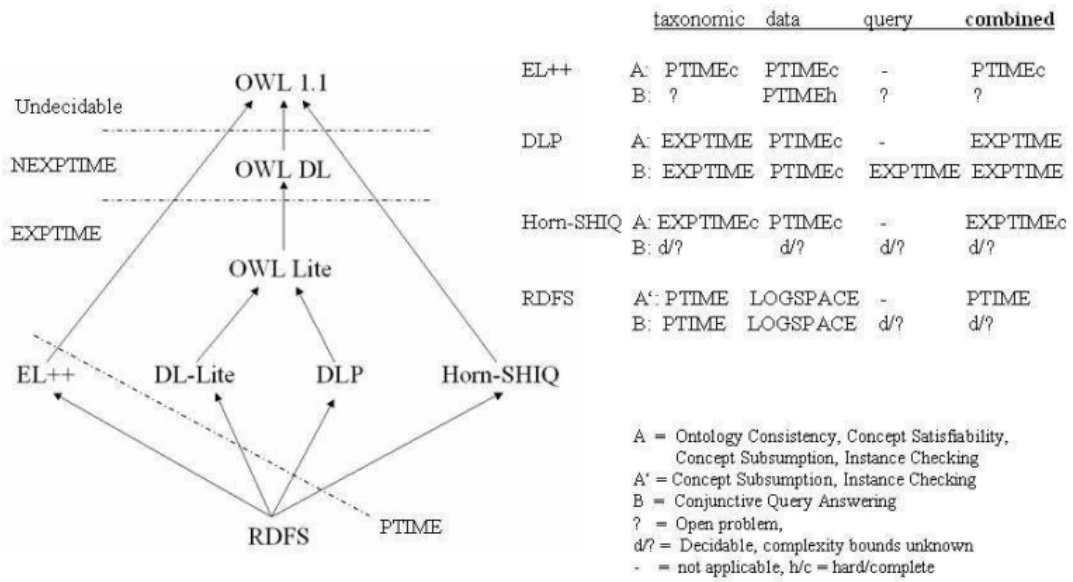


Рис. 1.4 - Розв’язні фрагменти OWL ([2])

Ще один важливий висновок щодо онтологій OWL приводиться з урахуванням логічного слідування онтології: онтологія O_1 слідує іншій O2, O1 | = O2, якщо всі інтерпретації, що задовільнять O_1, також задовільнять O_2 в значенні DL. Для обох різновидів мови OWL-DL (SHOIN (D)) та OWL-Lite (SHIF (D)) перевірка логічного слідування онтології може бути поліноміально зведена до перевірки виконуваності відповідних баз знань DL O_1, O_2 (перевірка послідовності онтологій), яка є розв'язуваною для обох різновидів.

1.4.4 Сервісний профіль

Онтологія профілю OWL-S використовується для описування того, що робить такий сервіс, і призначена для використання в основному з метою виявлення сервісу. Профіль сервісу OWL-S або підпис охоплює його функціональні параметри, наприклад, hasInput, hasOutput, precondition та effect(IOPEs), а також нефункціональні параметри, такі як serviceName, serviceCategory, qualityRating, textDescription та метадані (агент) про провайдера сервісу та інших відомих замовників. Просимо звернути увагу на те, що, на відміну від OWL-S 1.0, в OWL-S 1.1 визначаються параметри сервісу IOPE в

моделі процесу з унікальними посиланнями на ці визначення з профілю (див. Рис. 1.5).

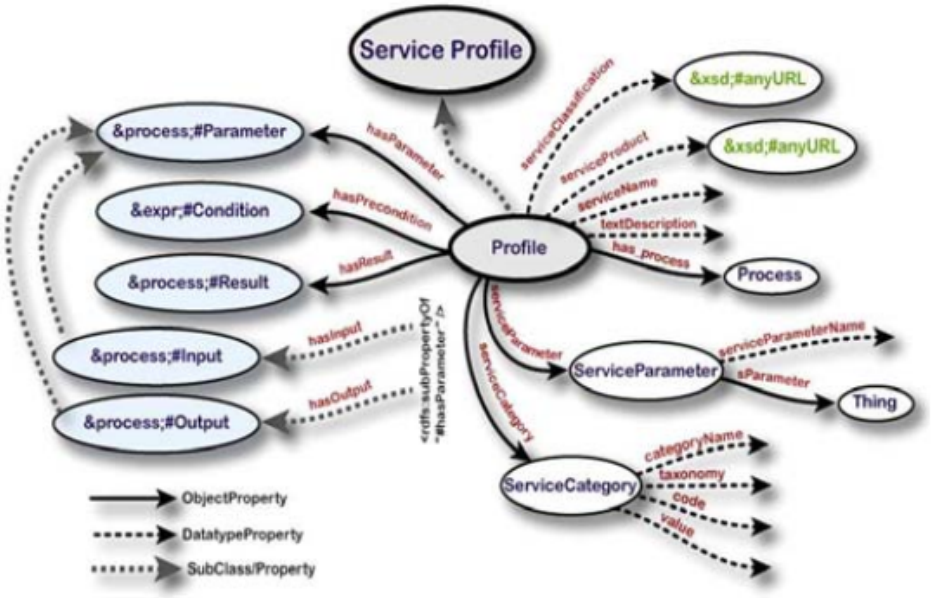


Рис. 1.5 - Структура профілю сервісу OWL-S

Входи та виходи відносяться до каналів даних, де проходять потоки даних між процесами. Передумови встановлюють факти предметної області (стану), які повинні бути підтверджені для виконання агентом сервісу. Ефекти характеризують факти, які стають підтвердженими, якщо сервіс успішно виконано в реальній предметній області (стані). У той час як семантика кожного вхідного та вихідного параметрів визначається як поняття OWL, формально зазначене в даній онтології, як правило, у розв’язному різновиді OWL-DL чи OWL-Lite, передумови та ефекти можуть бути виражені в будь-якій відповідній логікі (правилі), а саме KIF, PDDL та SWRL. Крім того, клас профілю може бути розділений на підкласи та тематики, тим самим підтримуючи створення таксономії профілю, які в свою чергу описують різні категорії сервісу. Приклад профілю семантичного веб-сервісу в OWL-S 1.1 наведено на Рисунку 1.6.

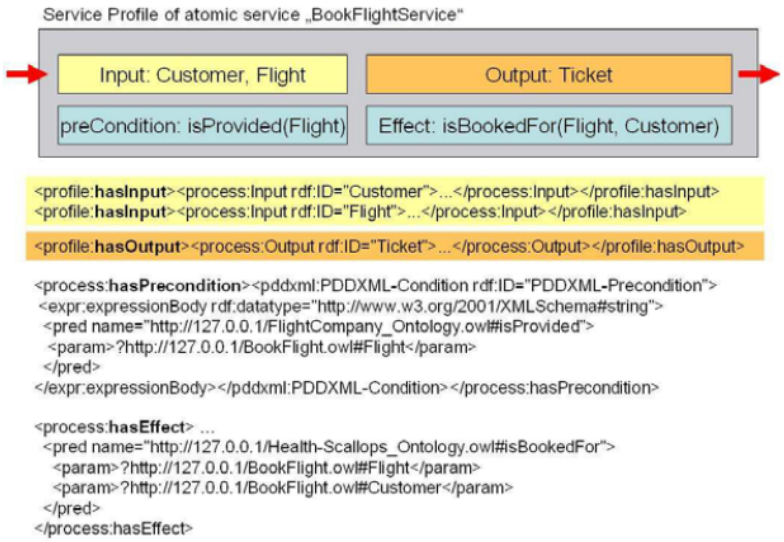


Рис. 1.6 - Приклад профілю сервісу OWL-S 1.1

1.4.5 Модель сервісного процесу

Модель процесу OWL-S описує структуру («хореографію та оркестровку») одного чи більше сервісів, тобто контрольоване введення в дію складових процесів з відповідним шаблоном комунікації. У OWL-S це охоплюється загальною підмножиною властивостей робочого потоку, таких як split+join, sequence, та choice (див. Рисунок 1.7).

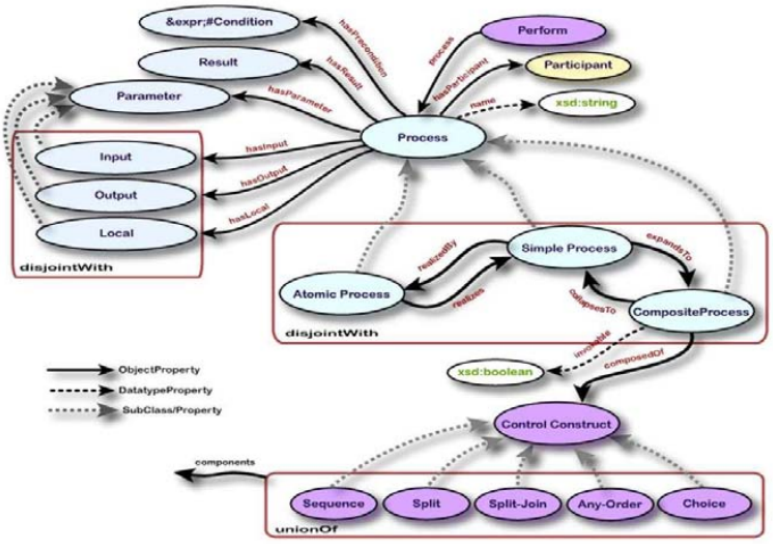


Рис. 1.7 - Модель сервісного процесу OWL-S

Вдаваючись у подробиці, процес в OWL-S може бути атомарним, простим або комплексним. Атомарний процес являє собою єдиний опис процесу типу «чорний ящик» з відкритим IOPE. Прості процеси надають засіб опису абстракцій сервісів чи процесів, які не мають специфічного зв'язку з реальним сервісом, таким чином, повинні бути реалізовані за допомогою атомарного процесу, наприклад, завдяки виявленню сервісу та динамічному зв'язуванні під час виконання, або має бути розширений у комплексний процес. Модель процесу сервісу OWL-S наведено на Рисунку 1.8.

Комплексні процеси – це ієрархічно визначені робочі потоки, що складаються з атомарних, простих та інших комплексних процесів. Ці технологічні процеси побудовані з використанням декількох різних операторів управління потоком, в тому числі Sequence, Unordered (lists), Choice, If-then-else, Iterate, Repeat-until, Repeat-while, Split, та Split+Join.

В OWL-S 1.1 модель процесу також визначає входи, виходи, передумови, і наслідки всіх процесів, що входять до комплексного сервісу, на які посилаються в профілях відповідних сервісів. Модель процесу OWL-S комплексного сервісу також може вказати, що його вихід дорівнює певному виходу одного з його під процесів кожного разу, коли створюється комплексний процес. Більше того, для комплексного процесу з конфігурацією управління послідовністю, вихід одного під процесу може бути визначено як вхід до іншої під процесу (зв'язування).

На жаль, семантика моделі процесу OWL-S залишається невизначеною в офіційних документах OWL-S. Хоча є пропозиції щодо уточнення цих семантик з точки зору, наприклад, обчислення ситуації [18] та логічної мови програмування GOLOG на основі цього обчислення [19].

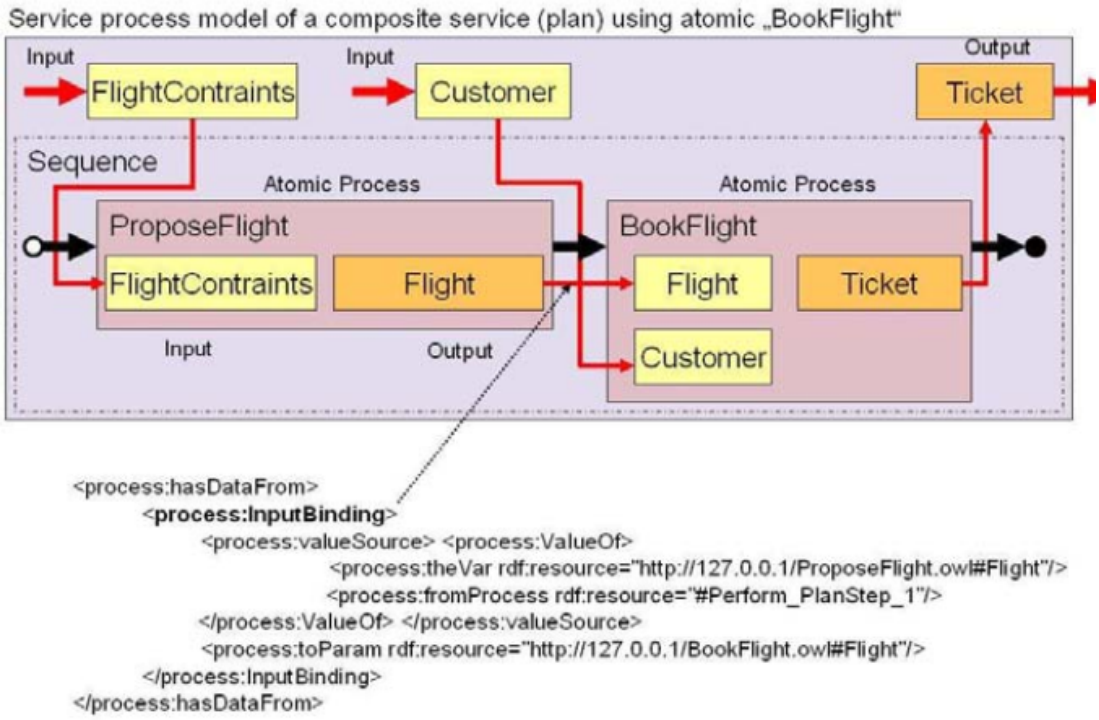


Рис. 1.8 - Приклад моделі сервісного процесу OWL-S

1.4.6 Прив’язка сервісу

Прив’язка даного опису сервісу OWL-S забезпечує прагматичне зв'язування між визначеннями логічного та XMLS-сервісу з метою сприяння виконанню сервісу. Така «прив’язка» сервісів OWL-S може бути, в принципі, довільною, але продемонстрована для прив’язки в WSDL, щоб прагматично підключити OWL-S до існуючого стандарту веб-сервісу (див. Рисунок 1.9).

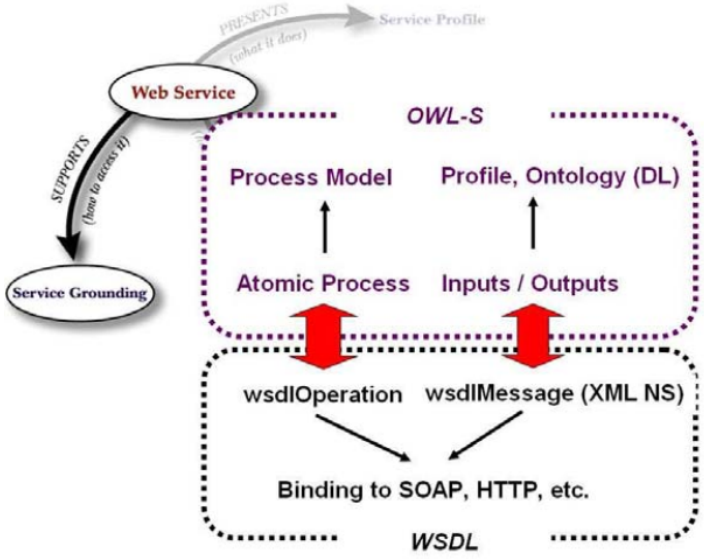


Рис. 1.9 - Прив’язка OWL-S в WSDL

Зокрема, модель процесу сервісу OWL-S поєднана з описом WSDL за допомогою тонкої (неповної) прив’язки: кожен атомарний процес відображається на операції WSDL, а властивості OWL-S, використані для представлення вхідних та вихідних даних, прив’язані з точки зору відповідних названих типів даних XML відповідних вхідних та вихідних повідомлень.

На відміну від OWL-S, WSDL не можна використовувати для вираження передумов або наслідків виконання сервісів. Будь-який атомарний або комплексний сервіс OWL-S із прив’язкою в WSDL виконується або безпосередньо за допомогою програми (служби), яка посилається у файлі WSDL, або за допомогою механізму BPEL, який обробляє основи WSDL простого або оркестрованого семантичних веб-сервісів.

1.5 Опис онтології WSML

Структура WSMO (онтологія моделювання веб-сервісів) забезпечує концептуальну модель та формальну мову WSML (мова веб-сервісів моделювання) для семантичного анотування веб-сервісів разом з базовою

реалізацією WSMX (середовище виконання веб-сервісу). Історично так склалося, що структура WSMO розвинулася на базі структури моделювання веб-сервісів (WSMF) в результаті декількох дослідницьких проектів, які фінансувалися Європейською Комісією, у сфері семантичних веб-сервісів, таких як DIP, ASG, Super, TripCom, KnowledgeWeb та SEKT в осередку проектів ESSI (Європейська ініціатива семантичних систем).

1.5.1 Стисла концептуальна модель WSMO

WSMO пропонує чотири ключові компоненти для моделювання різних аспектів семантичного веб-сервісу в WSML: онтології, цілі, сервіси та посередники. Цілі в репозиторіях визначають завдання, які може мати клієнт при пошуку відповідного веб-сервісу. Онтології WSMO в WSML забезпечують формальне логічне прив’язування інформації, що використовується всіма іншими компонентами моделювання. Посередники обходять проблеми взаємодії, які з’являються між усіма цими компонентами при даних (посередництво структури даних), протоколі (посередництво протоколів обміну повідомленнями) та рівні процесу (посередництво рівня предметної області), щоб «дозволити вільний зв’язок між веб-сервісами, цілями (запитами) та онтологіями». Кожному з цих компонентів, названих елементами верхнього рівня концептуальної моделі WSMO, за рекомендацією можуть бути приписані нефункціональні властивості, які слід взяти зі стандарту метаданих DublinCore. Нижче наведено докладні приклади сервісів та цілей.

Мова WSML спеціально розроблена для опису семантичного веб-сервісу з урахуванням його функціональності (сервісні можливості), імпортованих онтологій в WSML, та інтерфейсу, через який вона може бути доступна для «оркестровки та хореографії». Формальна семантика елементів в описуванні цілей та сервісів (передумови та пост умови) визначається як логічні аксіоми та обмеження в онтологіях за допомогою одного з п’яти різновидів WSML.

1.5.2 Різновиди WSML

Синтаксис WSML в основному випливає з F-логіки, розширеної більш докладними ключовими словами, та змінюється залежно від логічних виразів, які дозволяють описувати семантику сервісу та елементи описування цілей. WSML має зручний для сприйняття людиною синтаксис, а також синтаксис XML та RDF для обміну між машинами. Мова подається в різних варіантах, кожен з яких ґрунтується на певній логікі з різною виразністю та складністю обчислювань, а саме: DL (WSML-DL), LP (WSML-Flight, WSML-Rule) та немонотонічна логіка (WSMLFull) (див. Рисунок 1.10).

1. **WSML-Core** відповідає перетину DL (SHIQ (D)) та логіки Хорна з підтримкою типу даних, і ґрунтується на дискрипційному логічному програмуванні (DLP) [9]. Семантика WSML-Core визначається через пряме відображення безфункціональної логіки Хорна із застосуванням виразів WSML-Core під модельно-теоретичною семантикою та класичним послідовним зв'язком з мовою програмування PL1. WSML-Core повністю сумісна з підмножиною OWL-DL, і розширюється в напрямку DL (WSML-DL) та LP (WSML-Flight).

2. **WSML-DL** – це розв'язний DL різновид F-логіки, який розширює WSML-Core до SHIQ (D), що підпорядковується SHIF (D) на основі OWL-Lite, і підпорядковується SHOIN (D) на основі OWL-DL. Теоретико-модельна семантика WSML-DL узагальнює семантику WSML-Core і визначається шляхом відображення з рівністю на безфункціональній мові PL1. WSML-DL надає лише незначне моделювання обмежень (відсутні обмеження в замкненому світі) і не має довільних правил.

3. **WSML-Flight** – це розв'язний різновид мови Datalog F-логіки (без функціональні, нерекурсивні та DL-безпечні правила). Його примітивні модулі дозволяють вказувати різні аспекти атрибутів, такі як обмеження характеристик

та обмеження цілісності (за допомогою вбудованих модулів), а безпечні правила Datalog, що поширюються на нерівність та (локально) стратифіковане заперечення, допускають ефективні розв’язні обґрунтування. Іншими словами, в WSML-Flight, поняття, екземпляри та атрибути інтерпретуються як об'єкти в F-логіці з (немонотонним) стандартним запереченням за семантики ідеальної моделі [22] локально стратифікованих програм F-логіки з основним логічним слідуванням.

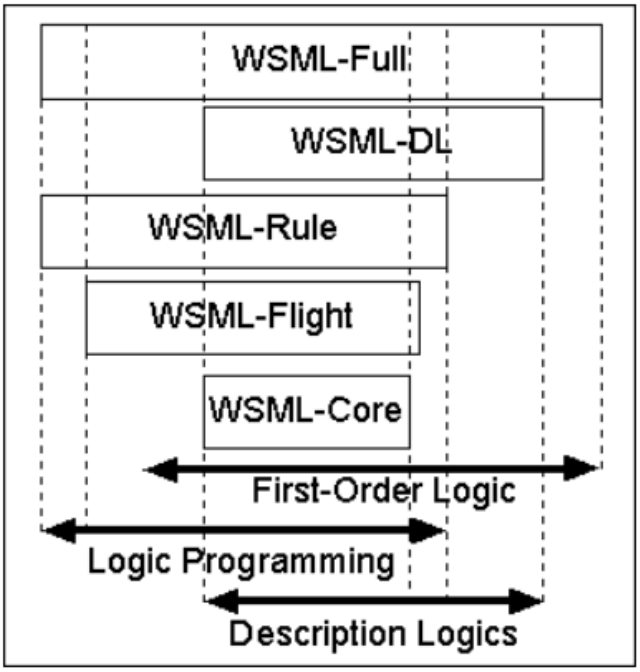


Рис. 1.10 - Різновиди мови WSML

4. **WSML-Rule** розширює WSML-Flight до повноцінної мови програмування LP, тобто з функціональними символами, і допускає довільні, небезпечні правила з нерівністю і стандартним запереченням. Цей різновид також забезпечує мета-моделювання, таке як обробка понять як екземплярів, але не має екзистенціалів, класичного (монотонного) заперечення та аргументації рівності. Семантика WSML-Rule визначається в так самий спосіб, як і в WSML-Flight, але через відображення на повнофункціональному LP, тобто на фрагменті Хорна F-логіки, розширеного з нерівністю та стандартним запереченням за добре

обґрунтованої семантики [26] в збірнику правил, а не через відображення на Datalog. Якщо коротко, то семантика WSML-Rule базується на добре обґрунтованій семантиці, яка застосовується до фрагменту LP F-Logic [27].

5. **WSML-Full** має об'єднати парадигми DL і LP як супер множина логіки першого порядку (FOL) з немонотонними розширеннями для підтримки немонотонного заперечення WSMLRule за допомогою стандартної логіки, обмежувальної або автоепістемічної логіки. Однак ні синтаксис, ні семантика WSML-Full все-таки не визначені повністю.

Загалом, описування семантики сервісу і запиту (цілі) в WSML структуроване на частини можливостей сервісу, інтерфейсу сервісу, що використовується для «оркестровки та хореографії», а також спільних змінних.

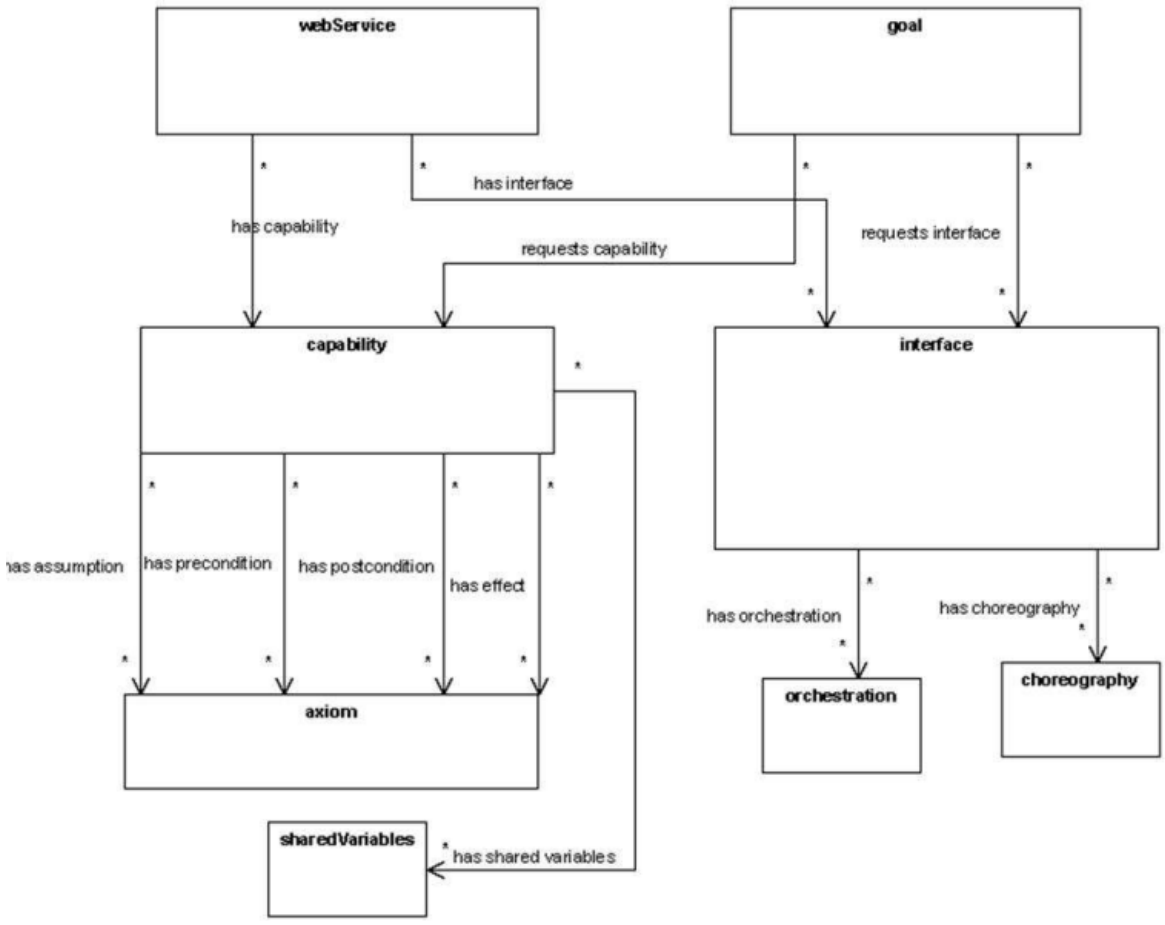


Рис. 1.11 - Опис сервісу WSML та опис цілі

1.5.3 Ціль

Як і в OWL-S, ціль у WSMO являє собою бажаний сервіс WSML, який позначається спеціальним ключовим словом «ціль» (goal) замість слова «веб-сервіс» перед описом сервісу. Ціль – це бажаний стан, який можна описати за допомогою онтології (предметної області). Така онтологія надає основний словник для визначення формальної семантики параметрів сервісу та правил переходу (TBox), та набір екземплярів концепцій і ролей (ABox), які можуть змінювати свої значення з однієї предметної області в іншу. Вона також визначає можливі права доступу до читання та запису екземплярів та їх «прив'язку». Стан – це динамічний набір екземплярів концепцій, відношень та функцій онтології даного стану на певний момент часу. Інтерпретація цілі (та сервісу) в WSML не є унікальною: користувач може захотіти виражати всі, або лише деякі цілі, що містяться в описаному наборі запиту [15].

```
namespace { _"http://example.org/goals#", dc _"http://purl.org/dc/elements/1.1#",
            tr _"http://example.org/tripReservationOntology",
            wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
            loc _"http://www.wsmo.org/ontologies/locationOntology#"}
goal _"http://example.org/havingATicketReservationInnsbruckVenice"
    importsOntology { _"http://example.org/tripReservationOntology",
                      _"http://www.wsmo.org/ontologies/locationOntology"}
    capability
        postcondition definedBy
            ?reservation[ reservationHolder hasValue ?reservationHolder,
                          Item hasValue ?ticket ]
                      memberOf tr#reservation and
            ?ticket[ trip hasValue ?trip ] memberOf tr#ticket and
            ?trip [ origin hasValue loc#innsbruck, destination hasValue loc#venice ]
                memberOf tr#trip.
```

Рис. 1.12 - Приклад сервісного запиту (цілі) у WSML

На Рисунку 1.12 наведено приклад цілі в WSML для пошуку сервісу, який внаслідок свого виконання, пропонує забронювати квиток на бажану поїздку. В

цьому випадку, єдиним елементом можливостей, яким зацікавлений користувач, є поступова бажаної послуги.

1.5.4 Можливість сервісу

Можливість сервісу WSML описує функціональність сервісу на основі станів за передумови (умови на інформаційному просторі), пост умови (результат виконання послуги, наданої користувачеві), припущення (умови в предметній області, яких потрібно дотримуватись перед виконанням послуги), та наслідку (як виконання змінює предметну область). Приблизно кажучи, можливість сервісу WSML складається з посилань на логічні вирази у різновиді мови WSML, названі за обсягом (передумова, пост умова, припущення, наслідок, можливість), який вони мають намір описати. Вона також вказує на нефункціональні властивості та загальновизнані спільні змінні універсальної квантифікації (з можливостями сервісу щодо області застосування), для яких логічна кон'юнкція передумови та припущення є результатом логічної кон'юнкції пост умови та наслідку.

Рисунок 1.13 наводить приклад можливостей веб-сервісу, зазначених у WSML. Цей приклад сервісу пропонує інформацію про поїздки, починаючи з Австрії, та видає запит на ім'я особи та реквізити кредитної картки для оформлення бронювання. Припущення полягає в тому, що інформація про кредитну картку, надана замовником, повинна вказувати на дійсну кредитну картку типу PlasticBuy або GoldenCard. У постумові зазначено, що бронювання, яке містить інформацію про квиток на бажану поїздку, та власник бронювання є результатом успішного виконання веб-сервісу. Нарешті, наслідком в предметній області є те, що з кредитної картки знімається вартість квитка.

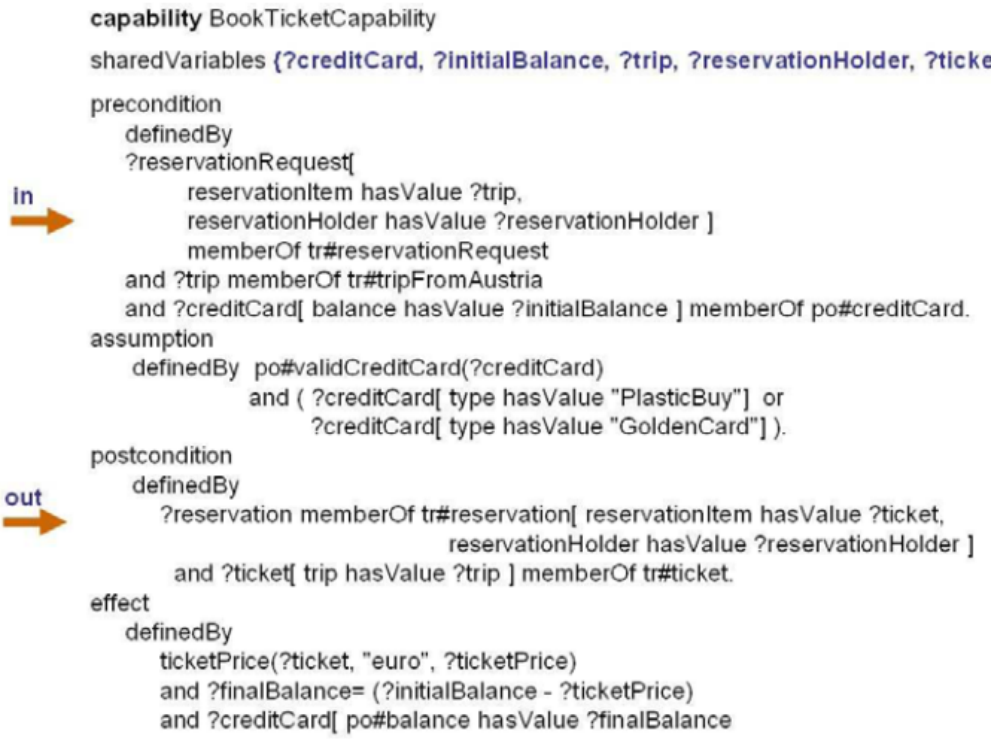


Рис. 1.13 - Приклад можливості сервісу в WSML

1.5.5 Інтерфейс сервісу

Інтерфейс сервісу WSML містить опис того, як загальна функціональність веб-сервісу досягається шляхом співпраці різної веб-сервісів провайдерів (оркестровка) та опис моделі комунікації, яка дозволяє користуватися функціональними можливостями веб-сервісу (хореографія). Опис хореографії складається з двох частин: переходи станів та захищені переходи. Як згадувалося вище, стан представлений онтологією WSMO, в той час як захищені переходи – є правилами «якщо-тоді» (if-then), що вказують на умовні переходи між станами в абстрактному просторі станів.

На Рис. 1.14 наведено приклад сервісного інтерфейсу з хореографією, та правило захищеного переходу, яке вимагає дотримуватись наступного: якщо існує екземпляр запиту на бронювання (воно вже отримано, оскільки відповідне поняття в онтології стану на даний момент має режим «in») з запитом на поїздку,

починаючи з Австрії, та існує екземпляр квитка для бажаної поїздки в сховищі екземплярів веб-сервісу, тоді створюється тимчасове бронювання цього квитка.

```
interface BookTicketInterface
        importsOntology _http://www.example.org/BookTicketInterfaceOntology
        choreography BookTicketChoreography
        orchestration BookTicketOrchestration

choreography BookTicketChoreography
        state _"http://example.org/BookTicketInterfaceOntology"
        guardedTransitions BookTicketChoreographyTransitionRule

guardedTransitions BookTicketChoreographyTransitionRule
if (  reservationRequestInstance [ reservationItem hasValue ?trip,
                                   reservationHolder hasValue ?reservationHolder ]
     memberOf bti#reservationRequest
     and ?trip memberOf tr#tripFromAustria
     and ticketInstance[ trip hasValue ?trip, recordLocatorNumber hasValue ?rln ]
         memberOf tr#ticket )
then temporaryReservationInstance[ reservationItem hasValue ticketInstance,
                                   reservationHolder hasValue ?reservationHolder ]
     memberOf bti#temporaryReservation
```

Рис. 1.14 - Приклад сервісного інтерфейсу WSML

1.6 Монолітне описання сервісу на базі описової логіки

Як зазначалося вище, альтернатива формальному визначенню функціональної семантики веб-сервісу, незалежного від будь-яких структурованих форм описання сервісів, таких як OWLS, SAWSDL або WSML, – це чистий підхід на базі описової логіки DL: абстрактна семантика сервісу визначається через відповідний набір концепцій та рольових аксіом в даній описовій логіці. Створення будь-якого екземпляру цієї концепції сервісу відповідає конкретному сервісу з конкретними сервісними властивостями. Тобто розширення S_Iсервісної концепції S, що представляє собою абстрактну послугу, яка описується в інтерпретації I поняття в певному домені, містить всі сервісні екземпляри, які провайдер S бажає прийняти для укладання угоди з потенційним

замовником на *S*. На Рис. 1.15 наведено приклад опису монолітного абстрактного сервісу на базі DL та можливі сервісні екземпляри [8].

У цьому прикладі функціональна семантика або можливість абстрактного веб-сервісу *S* описується набором *DS*двох аксіом концепції DL: концепція сервісу *S* для відправки товарів вагою менше 50кг з міст Великобританії до міст Німеччини; концепція *Shipping*(використовується для визначення *S*), яка гарантує, що екземпляри *S* вказують точно одне місце для відвантаження та місця доставки. Семантичні відносини між такими монолітно охарактеризованими семантиками сервісу може бути повністю визначені в межах логічного формалізму, тобто за допомогою логічного припущення на базі DL. Для більш детального розгляду цієї теми ми звертаємось до [8].

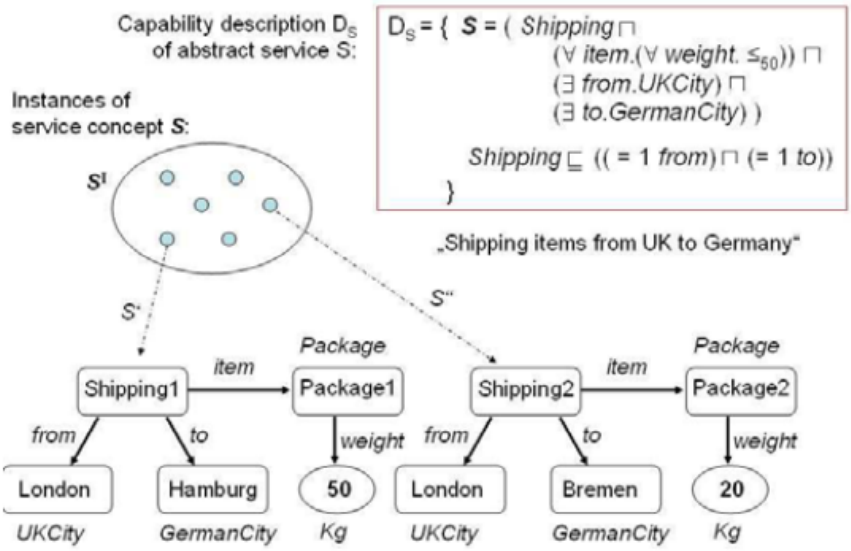


Рис. 1.15 - Приклад опису монолітного семантичного сервісу на базі DL

1.6 Критика

Головна критика семантичних веб-сервісів (SWS) варіюється від обмежень запропонованої моделі через відсутність відповідних засобів координації сервісів та підтримки програмного забезпечення до узаконення області досліджень в цілому. Як наслідок, технологія SWS все ще виглядає надто незрілою для того,

щоб бути прийнятою і веб-користувачами, і розробниками на практиці, а також промисловою галуззю для своїх комерційних цілей у великих масштабах.

1.6.1 Необхідність формальної семантика сервісу

Деякі сучасні критики технології SWS виступають проти значущості заявлених переваг технології для практичного застосування веб-сервісів в цілому. Ключове обґрунтування цього аргументу пов’язано з загальною критикою семантичних веб-технологій. Насправді, потреба у формальній логічній семантиці, визначеної для веб-сервісів, із людино-орієнтованим застосуванням, часто піддається сумніву: абсолютно незрозуміло, чи повна відсутність формальної семантики сервісу виявиться досить незначною або досить критичною, для яких саме прикладних програм для звичайного веб-користувача на практиці, та в яких масштабах.

1.6.2 Де знаходяться всі семантичні веб-сервіси?

Ще одне цікаве питання стосується реальної дійсності технології SWS в процесі використання. Згідно з недавнім опитуванням публічно доступних семантичних веб-сервісів в поверхневій мережі [17] виявлено, що не більше ніж близько 1500 індексованих семантичних сервісів у OWL-S, WSML, WSDL-S або SAWSDL доступні в Інтернеті, з яких лише близько ста розташовані поза межами спеціальних тестових зборів, такі як OWLS-TC. Хоча ми очікуємо, що більшість семантичних веб-сервісів зберігаються в приватних репозиторіях та на сайтах глибинного інтернету [10], це, безумовно, не відображає наполегливих дослідницьких зусиль у сфері SWS по всьому світі.

Звичайно, можна стверджувати, що в цьому це немає нічого дивного у двох випадках. По-перше, технологія SWS є нерозвиненою (з нещодавно прийнятим стандартом, а саме SAWSDL), яка все ще надає незначну загальну платформу, що підтримує своє використання кінцевими користувачами. Хоча це, безумовно, правда, іншою стороною цього аргументу є те, що масове дослідження та

розвиток навколишнього світу мали б створити значну кількість описів навіть публічно доступних семантичних веб-сервісів протягом останніх півтора десятка років.

По-друге, можна стверджувати, що незрозуміло, чи є поверхнева мережа та академічні публікації правильним місцем для пошуку описів семантичних веб-сервісів, оскільки багато з них призначені для внутрішнього або між корпоративного використання, а не доступні для громадськості. Хоча це є однією з можливих причин низьких даних, зазначених вище, та це свідчить про певну відсутність доступності для загального веб-користувача на сьогоднішній день.

1.6.3 Де прості у використанні інструменти SWS для широкого загалу?

Як і розробка семантичних веб-додатків загалом, крім прототипів та систем проектів навряд чи легко використовувати програмне забезпечення з полиць, доступних для звичайних користувачів з метою розробки, повторного використання та обміну власними семантичними веб-сервісами – що може перешкоджати нинішньому злитті поля з Web 2.0 в так званий сервіс Web 3.0 на практиці.

1.6.4 Як ефективно координувати семантичні веб-сервіси?

Незважаючи на величезний прогрес в європейських та національних науково-дослідних проектах, таких як DIP, Super, CASCOM, Scallops та SmartWeb,існує достатньо місця для подальшого вивчення характеристик, потенціалу та обмежень координації SWS як в теорії, так і на практиці. У поданому документі «Про виклик для семантичних веб-сервісів» (SemanticWebServicesChallenge)викладено спробу якісно виміряти мінімальну кількість програм, необхідних для адаптації семантики даних систем до нових сервісів. Це підтверджує, що повна автоматизація компонування раніше невідомих сервісів є неможливим – так би мовити, своєрідний Священний Грааль сучасних семантичних технологій. Крім того, порівняльну оцінку розроблених

інструментів для виявлення SWS на даний час важко виконати, якщо не неможливо, оскільки все ще відсутні необхідні великомасштабні вибіркові тестові збори сервісу навіть для стандартного SAWSDL.

По цій причині немає доступних на практиці великомасштабних експериментальних результатів щодо масштабованості запропонованих засобів координації сервісів.

Окрім проблеми масштабованості та ефективності виявлення та створення SWS, ще однією відкритою проблемою координації SWS в цілому є збереження конфіденційності. Хоча існує досить багато підходів до збереження конфіденційності даних користувачів для кожного індивідуального процесу координації (виявлення, створення та переговори), немає інтегрованого підходу, який дозволяє послідовно забезпечувати координаційну роботу SWS.

1.7 Висновки

У цьому розділі коротко представлено відомі моделі описання сервісу в семантичній мережі разом з деякими головними критичними зауваженнями в даній сфері. В цілому, комплексне, наглядне дослідження та розвиток семантичного Інтернету досягли вражаючих результатів як у теорії, так і на практиці протягом декількох років з їх появою в 2005 році. Хоча ми виявили кілька основних прогалин в все ще недосить розвиненій технології семантичних веб-сервісів, поточна конвергенція семантичного вебу, Web 2.0 і так званий сервіс Web 3.0 показує свій потенціал для майбутніх сервісних веб-додатків.

2 NEO4J ДЛЯ ПОБУДОВИ СЕМАНТИЧНОГО МІКРО СЕРВІСУ ДЛЯ РОБОТИ З ГРАФОВОЮ БАЗОЮ ДАНИХ

Існує дві характеристики RDF сховищ. Перша, і між іншим, найвагоміша, зберігати дані та отримувати за допомогою графу. Друга полягає в тому, що вони є семантичними, це досить вагомий спосіб сказати, що вони можуть зберігати не тільки дані, але й чіткі описи значення цих даних. RDF бази часто посилаються на ці описи, використовуючи онтології. Онтологія - це такий опис домену, який зазвичай включає в себе словник термінів та деяку специфікацію взаємодії цих термінів, накладаючи структуру на дані для певного домену. Це також можна назвати схемою. Взагалі обидва терміни, і схема, і онтологія є взаємозамінні.

Включення семантики до ваших даних за допомогою онтології дозволить забезпечити обмін даними. В певних сценаріях ви можете використати свою онтологію, щоб виконувати базові вибірки зі своїх даних, щоб отримати нові факти з існуючих. Ще одним подібним способом використанням семантики може бути перевірка даних на конкретну послідовність для домену. Значення конкретної послідовності для домену, сильно відрізняється від обмежень цілісності, що притаманно реляційним базам даних. Насправді йдеться про визначення наступного правила: якщо одиниця даних прив’язана до сутності ПАЦІЄНТА за допомогою відношення МАЄ_АНАЛІЗ, то це є сутність ДАНІ_ПАЦІЄНТА.

Спочатку подивимося, як виразити такі перевірки послідовності, потім як їх можна застосувати:

- контроль додавання даних в базу даних на рівні операцій, перевіряючи, перш ніж застосовувати, що набір даних, який передано, залишає базу в консистентному стані та як і раніше буде відповідати семантиці схеми, в іншому випадку виконати скасувати операції.

- використовуючи онтологію в зовнішньому інтерфейсі для гарантування, що через такий інтерфейс додаються лише дані, які залишать систему консистентною.
- виконання перевірки послідовності на наборах даних для визначення невідповідностей. Хоча попередні два можна вважати переважаючими, цей підхід буде корекційним. Прикладом цього може служити створення набору даних (графа в нашому випадку) шляхом надходження даних з різних джерел, і перевірка, чи вони поєднуються послідовно відповідно до деякої попередньо визначеної семантики схеми.

Наразі, головна ціль полягає в тому, щоб дослідити, як семантичне сховище може бути перенесено на таку не семантичну базу даних, як Neo4j.Для цього потрібен простий спосіб зберігання інформації про схему поряд з звичайними даними таким чином, щоб обидва могли бути одночасно отримані разом. Нам також знадобиться словник для вираження семантики в Neo4j, і для цього виберемо деякі з примітивів в OWL і RDFs, щоб виробити базову мову визначення онтології.

2.1 Опис онтології в стилі RDF/OWL на базі Neo4j

W3C рекомендує використовувати RDFs та OWL для визначення онтологій. Для нашого прикладу будемо використовувати мову, натхненну цими двома, в якій будемо використовувати такі елементи як owl:Class, owl:ObjectProperty, owl:DatatypeProperty, rdfs:domain, rdfs:range, тощо.

Почнемо з короткого пояснення основних примітивів в нашій мові визначення онтології: клас визначає категорію, яка в Neo4j відповідає міткі вузла. DatatypeProperty описує атрибут у Neo4j, а ObjectProperty описує залежності. Властивості домену та діапазону в онтології використовуються для подальшого опису визначень ObjectProperty і DatatypeProperty. У випадку

ObjectProperty домен та діапазон визначають джерело та цільові класи вузлів, з'єднаних з екземплярами ObjectProperty.

Аналогічним чином, у випадку DatatypeProperty у домені буде вказано клас вузлів, що зберігають значення для такої властивості та діапазону, коли присутність може бути використана для вказування типу даних XMLSchema значень властивостей. Зауважте, що оскільки модель власності графів приймає властивості у відношенні, доменом DatatypeProperty може бути ObjectProperty, який не є дійсним для RDF.

Всі елементи в схемі ідентифікуються за URI, що слідують за стилем RDF, навіть якщо це не є суто необхідним для нашого експерименту. Безумовно, це посилання на те, як кожен елемент в онтології фактично використовується в даних Neo4j, і це ім'я зберігається в властивості мітки.

Завантаживши Neo4j, можна отримати доступ до бази даних про фільми. Використаємо цю базу даних для свого експерименту. На рис 2.1зображено фрагмент онтології, яку було створено для цього (повний лістинг в Додатку 1). Можна побачити клас Person, ім'я DatatypeProperty і ACTED_IN ObjectProperty, визначені як описано в попередніх абзацах.

```
// A Class definition (a node label in Neo4j)
(
    person_class:Class {
        uri:'http://neo4j.com/voc/movies#Person',
        label:'Person',
        comment:'Individual involved in the film industry'
    }
)

// A DatatypeProperty definition (a property in Neo4j)
(   name_dtp:DatatypeProperty {
        uri:'http://neo4j.com/voc/movies#name',
        label:'name',
        comment :'A person\'s name'
    }
),

(name_dtp)-[:DOMAIN]->(person_class)

// An ObjectProperty definition (a relationship in Neo4j)
(   actedin_op:ObjectProperty {
        uri:'http://neo4j.com/voc/movies#ACTED_IN',
        label:'ACTED_IN',
        comment:'Actor had a role in film'
    }
),

(person_class)<-[:DOMAIN]-(actedin_op)-[:RANGE]->(movie_class)
```

Рис 2.1 - Фрагмент онтології в синтаксисі Cypher

Запустивши цей код в редакторі Neo4j, отримаємо граф, зображений на Рис 2. 2.

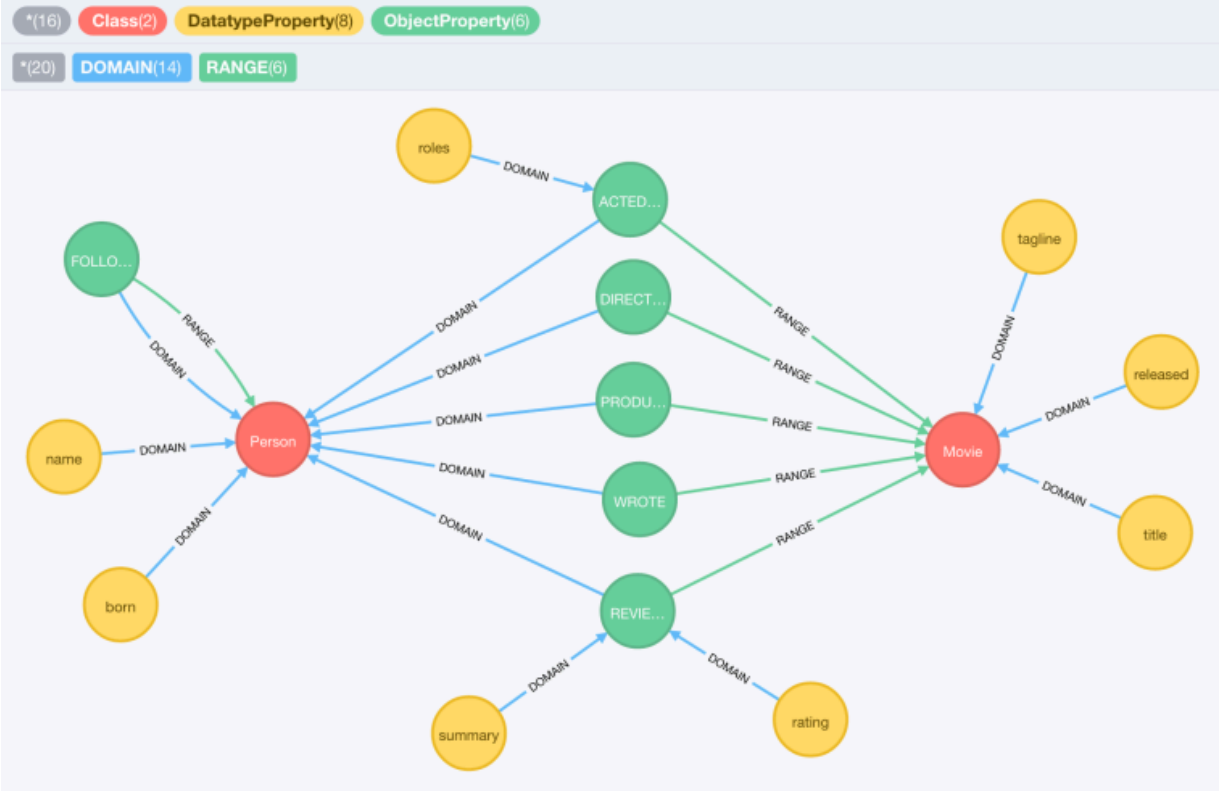


Рис. 2. 2 - Структура бази даних кінофільмів

Можливо, вам захочеться написати свою онтологію використовуючи Cypher, як це вже було зроблено, але це також можливо, що якщо у вас є деякі існуючі онтології OWL або RDFS (ось версія OWL з однаковою онтологією фільмів), вам буде потрібний загальний спосіб перевести їх в онтології Neo4j, як і попередній, і саме це зберігає процедура Neo4j. Таким чином, альтернатива запуску попереднього Cypher може полягати у розгортанні та запуску цієї збереженої процедури.

Фрагмент онтології на OWL зображено на Рис. 2.3.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY mov  "http://neo4j.com/voc/movies#" >
    <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
  ]>

<rdf:RDF
  xmlns     = "http://neo4j.com/voc/movies#"
  xmlns:mov = "hhttp://neo4j.com/voc/movies#"
  xml:base  = "http://neo4j.com/voc/movies#"
  xmlns:owl = "http://www.w3.org/2002/07/owl#"
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">

  <owl:Ontology rdf:about="">
    <rdfs:comment>A basic OWL ontology for Neo4j's movie database</rdfs:comment>
    <rdfs:comment>Simple ontology providing basic vocabulary and domain+range axioms
  for the movie database.</rdfs:comment>
    <rdfs:label>Neo4j's Movie Ontology</rdfs:label>
  </owl:Ontology>

  <owl:Class rdf:ID="Person">
    <rdfs:label xml:lang="en">Person</rdfs:label>
  <rdfs:comment xml:lang="en">Individual involved in the film industry</rdfs:comment>
  </owl:Class>

  <owl:Class rdf:ID="Movie">
    <rdfs:label xml:lang="en">Movie</rdfs:label>
  <rdfs:comment xml:lang="en">A film</rdfs:comment>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="name">
    <rdfs:label xml:lang="en">name</rdfs:label>
  <rdfs:comment xml:lang="en">A person's name</rdfs:comment>
    <rdfs:domain rdf:resource="#Person" />
  </owl:DatatypeProperty>
```

Рис 2.3 - Фрагмент онтології для бази кінофільмів на OWL

2.2 Перевірка консистентності даних, використовуючи Cypher

Якщо ми хочемо використовувати мову для опису онтологій, окрім як для створення орієнтованих графів, то потрібно буде впровадити механізми використання схем, визначених за допомогою цієї мови. Гарною новиною э те, що код буде один і той же, оскільки він працює для визначення будь якої схеми. Давайте подивимось, що це точно означає на прикладі. Правило, яке перевіряє послідовне використання зв'язків на графіку, може бути записане в Cypher наступним чином:

```
// ObjectProperty domain semantics check
MATCH (n:Class)<-[:DOMAIN]-(p:ObjectProperty)

WITH n.uri as class, n.label as classLabel, p.uri as prop, p.label as propLabel

MATCH (x)-[r]->() WHERE type(r)=propLabel AND NOT classLabel in Labels(x)

RETURN id(x) AS nodeUID, 'domain of ' + propLabel + ' [' + prop + ']' AS `check failed`,

'Node labels: (' + reduce(s = '', l IN Labels(x) | s + ' ' + l) + ') should include ' +
classLabel AS extraInfo

```

Рис 2.4 - Фрагмент коду на семантичну перевірку відношення між нодами

Цей запит сканує дані у вашій БД на наявність хибних відношень відповідно до онтології. Якщо у онтології ми стверджуємо, що ACTED_IN - це зв'язок між особою та фільмом, це правило знайде ситуацію, коли це не відповідає дійсності. Оскільки методика Cypher для визначення схеми надихається RDFS та OWL, має сенс слідувати їхній стандартній семантиці. Cypher працює на відношеннях DOMAIN. Поняття DOMAIN, якщо воно визначається між об'єктом ObjectProperty p та класом C, говорить, що будь-який вузол в наборі даних, який має значення для цього конкретного властивості p, є екземпляром класу C. Так, наприклад, коли я заявляю (actedin_op) - [: DOMAIN] -> (person_class), я маю на увазі, що Особи є предметом предиката ACTED_IN, або іншими словами, якщо вузол підключено до іншого вузла через зв'язок ACTED_IN, то він повинен мати позначку як: особа, оскільки в фільмі може виступати лише людина.

Ось інший скрипт Cypher дуже схожий на попередній, за винятком того, що він застосовується замість атрибутів відносини.

```
// DatatypeProperties on ObjectProperty domain semantics meta-rule (property graph specific. attributes on relationships)
MATCH (r:ObjectProperty)<-[:DOMAIN]-(p:DatatypeProperty)

WITH r.uri as rel, r.label as relLabel, p.uri as prop, p.label as propLabel

MATCH ()-[r]->() WHERE r[propLabel] IS NOT NULL AND relLabel<>type(r)

RETURN id(r) AS relUID, 'domain of ' + propLabel + ' [' + prop + ']' AS `check failed`, 'Rel type: ' + type(r) + ' but should be ' + relLabel AS extraInfo
```

Рис. 2.5 - Фрагмент коду на семантичну перевірку приналежності властивостей

Якщо запустити цей запит у базі даних фільмів, то не отримаємо жодних результатів, оскільки дані на графіку співпадають з онтологією: у фільмах працюють лише особи, у фільмах - лише назви, лише люди мають дату народження тощо. Але ми можемо спробувати викликати помилку, вставивши такий вузол:

MATCH (:Person {name:'DemiMoore'})-[r:ACTED_IN]->(:Movie {title:'A FewGoodMen'})
SET r.rating = 88

Це неправильний запит, адже встановлюється значення рейтингу для відношення, в якому немає такого атрибуту. Якщо ми зараз перезапустимо попередній запит на перевірку послідовності, він повинен повернути помилку зображену на Рис. 2.6.

```
+======+=================================================+=======================================+
|relUID|check failed                                     |extraInfo                              |
+======+=================================================+=======================================+
|64560 |domain of rating [http://neo4j.com/voc/movies#rating]|Rel type: ACTED_IN but should be REVIEWED|
+------+-------------------------------------------------+---------------------------------------+
```

Рис. 2.6 - Логування помилки

Наша онтологія виявила, що рейтингова властивість відсутня в ACTED_IN. Давайте видалимо це поле наступним запитом в базу:

MATCH (:Person {name:'DemiMoore'})-[r:ACTED_IN]->(:Movie {title:'A FewGoodMen'})
REMOVE r.rating.

Використання мов з чітко визначеною семантикою є досить потужним інструментом, оскільки це дозволяє побудувати сервіси загального призначення,

засновані лише на мовних примітивах, які можуть бути багаторазово використовуваними між доменами.

2.3 Перевірка консистентності на рівні запитів

Ми вже використали нашу онтологію, щоб керувати виконанням мета-правил для перевірки послідовності набору даних після внесення деяких змін. Як було вже описано раніше, ми можемо зробити так, щоб транзакції виконувалися лише в тому випадку, якщо вони залишають наш графік у стабільному стані.

```
import sys
from py2neo import Graph

def executewithconsistencycheck(ccs, statement):
    res = 0
    tx = graph.cypher.begin()
    tx.append(statement)
    tx.process()
    for cc in ccs:
        tx.append(cc.cccypher)
        ccresults = tx.process()[-1]
        if (len(ccresults) > 0):
            tx.rollback()
            print 'Consistency Checks failed. Transaction rolled back\n', ccresults
            res = 1
            break   #break as soon as one check fails.

    if (res == 0):
        tx.commit()
        print 'Consistency Checks passed. Transaction committed'

graph = Graph()
# cache the consistency check meta-rules
ccs = graph.cypher.execute("MATCH (cc:ConsistencyCheck) RETURN cc.ccid AS ccid, cc.ccname AS ccname, cc.cccypher AS cccypher")
# run a consistency safe transaction
res = executewithconsistencycheck(ccs, sys.argv[1])
```

Рис. 2.7 – Скрипт перевірки консистентності на мові Python

Логіка перевірки послідовності знаходиться на стороні клієнта, що може виглядати дещо дивно, але тут мається на меті зробити все це ясним і зрозумілим, наскільки це можливо. Перевірка полягає в тому, щоб керувати цілим набором окремих мета-правил, визначеним у попередніх розділах по одному, і порушувати, якщо будь-який з них піднімає невідповідність будь-якого типу. Код вимагає, щоб мета-правила були доступні на сервері Neo4j, і це було зроблено, зберігаючи кожен з них окремо, як вузли ConsistencyCheck з кодом Cypher, що зберігається як властивість. Щось на зразок цього:

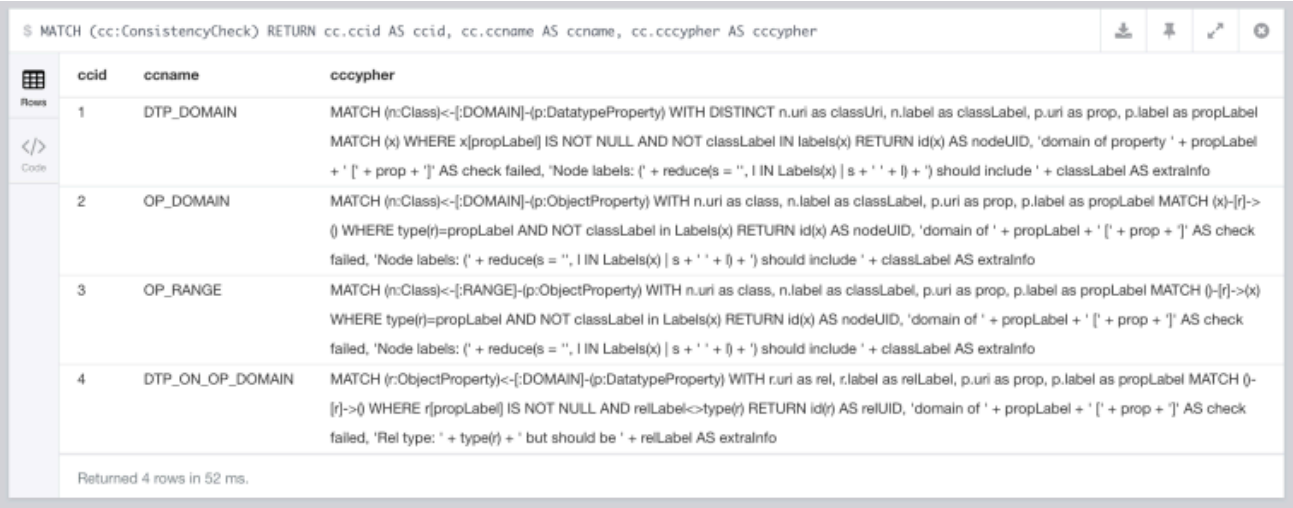
```
CREATE (ic:ConsistencyCheck { ccid:1, ccname: 'DTP_DOMAIN',
cccypher: 'MATCH (n:Class)<-[:DOMAIN]-(p:Datatyp ... '})
```

Тепер перевірка послідовності транзакцій може захопити всі мета-правила та кешувати їх цим простим запитом:

MATCH (cc:ConsistencyCheck)

RETURN cc.ccid AS ccid, cc.ccname AS ccname, cc.cccypher AS cccypher

, що повертає партію окремих мета-правил для запуску.



```
$ MATCH (cc:ConsistencyCheck) RETURN cc.ccid AS ccid, cc.ccname AS ccname, cc.cccypher AS cccypher
```

ccid	ccname	cccypher
1	DTP_DOMAIN	MATCH (n:Class)<-[:DOMAIN]-(p:DatatypeProperty) WITH DISTINCT n.uri as classUri, n.label as classLabel, p.uri as prop, p.label as propLabel MATCH (x) WHERE x[propLabel] IS NOT NULL AND NOT classLabel IN labels(x) RETURN id(x) AS nodeUID, 'domain of property ' + propLabel + ' [' + prop + ']' AS check failed, 'Node labels: (' + reduce(s = '', l IN Labels(x) | s + ' ' + l) + ') should include ' + classLabel AS extraInfo
2	OP_DOMAIN	MATCH (n:Class)<-[:DOMAIN]-(p:ObjectProperty) WITH n.uri as class, n.label as classLabel, p.uri as prop, p.label as propLabel MATCH (x)-[r]->() WHERE type(r)=propLabel AND NOT classLabel in Labels(x) RETURN id(x) AS nodeUID, 'domain of ' + propLabel + ' [' + prop + ']' AS check failed, 'Node labels: (' + reduce(s = '', l IN Labels(x) | s + ' ' + l) + ') should include ' + classLabel AS extraInfo
3	OP_RANGE	MATCH (n:Class)<-[:RANGE]-(p:ObjectProperty) WITH n.uri as class, n.label as classLabel, p.uri as prop, p.label as propLabel MATCH ()-[r]->(x) WHERE type(r)=propLabel AND NOT classLabel in Labels(x) RETURN id(x) AS nodeUID, 'domain of ' + propLabel + ' [' + prop + ']' AS check failed, 'Node labels: (' + reduce(s = '', l IN Labels(x) | s + ' ' + l) + ') should include ' + classLabel AS extraInfo
4	DTP_ON_OP_DOMAIN	MATCH (r:ObjectProperty)<-[:DOMAIN]-(p:DatatypeProperty) WITH r.uri as rel, r.label as relLabel, p.uri as prop, p.label as propLabel MATCH ()-[r]->() WHERE r[propLabel] IS NOT NULL AND relLabel<>type(r) RETURN id(r) AS relUID, 'domain of ' + propLabel + ' [' + prop + ']' AS check failed, 'Rel type: ' + type(r) + ' but should be ' + relLabel AS extraInfo

Returned 4 rows in 52 ms.

Рис. 2.8- Результат вибірки обов’язкових полів

Код може бути протестований за допомогою різних запитів на мові Cypher.

```
$ python transactional.py " CREATE (:Person { name: 'Marlon Brando'})-[:ACTED_IN]->(:Movie{ title: 'The Godfather'}) "
Consistency Checks passed. Transaction committed
```

Рис. 2.9 – Перевірка запиту на валідність

Тепер, якщо ми спробуємо оновити графік неконсистентними даними, мета-правила повинні спрацювати. Давайте спробуємо вставити вузол, позначений як Label з атрибутом, який призначений для використання вузлами Person:

```
$ python transactional.py " CREATE (:Thing { name: 'Marlon Brando'}) "
Consistency Checks failed. Transaction rolled back
   | nodeUID | check failed                                                  | extraInfo
---+---------+---------------------------------------------------------------+-----------------------------------------
 1 |    7231 | domain of property name [http://neo4j.com/voc/movies#name] | Node labels: ( Thing) should include Person
```

Рис 2.10 – Перевірка хибного запиту на валідність

Або зв'язати існуючого актора (:Person) з сутністю, що не є фільмом, через: ACTED_IN відношення:

```
$ python transactional.py " MATCH (mb:Person { name: 'Marlon Brando'}) CREATE (mb)-[:ACTED_IN]->(:Play { playTitle: 'The mousetrap'}) "
Consistency Checks failed. Transaction rolled back
   | nodeUID | check failed                                                | extraInfo
---+---------+-------------------------------------------------------------+-----------------------------------------
 1 |    7241 | domain of ACTED_IN [http://neo4j.com/voc/movies#ACTED_IN] | Node labels: ( Play) should include Movie
```

Рис. 2.11 – Результат хибного запиту

2.4 Перевірка консистентності на етапі генерації інтерфейсу користувача

Інший спосіб гарантувати цілісність даних у графової бази даних, коли він вручну задається через інтерфейс, полягає в тому, щоб створити загальний інтерфейс, керований вашою моделлю онтологій. Скажімо, наприклад, вам потрібно створити веб-форму для додання нових фільмів. Ну, ви можете отримати як структуру своєї форми, так і запит на введення даних за допомогою Cypher:

```
MATCH (c:Class { label: {className}})<-[:DOMAIN]-(prop:DatatypeProperty)
RETURN { cname: c.label, catts: collect( {att:prop.label})} asobject,
 'MERGE (:' + c.label + ' { uri:{uri}' + reduce(s = "", x IN collect(prop.label) | s + ',' +x + ':{' +
x + '}' ) +'})' ascypher
```

Знову ж таки, загальний запит, який працює на вашій схемі. Запит виконує параметр className. Якщо встановлено значення "Кіно", ви отримаєте у відповідь всю інформацію, необхідну для створення інтерфейсу вставки фільму. Ось фрагмент структури json, яку повернув Neo4j.

```
{
   "row": [
            {
              "cname": "Movie",
              "catts": [
                { "att": "tagline" },
                { "att": "title" },
                { "att": "released" }
              ]
            },
            "MERGE (:Movie { uri:{uri},tagline:{tagline},title:{title},released:{released}})"
          ]
}
```

Рис. 2.12 – Приклад результату запиту

2.5 Висновки

В даному розділі було показано, як на базі Neo4j можна побудувати семантичний мікро-сервіс. Показано варіанти опису онтології в стилі RDF/OWL. Описано приклади перевірки консистентності даних, використовуючи Cypher, перевірки консистентності на рівні запитів, що дало можливість перевіряти, чи вірно користувач подав відношення між класами. Також було описано підхід до перевірки консистентності на рівні генерації інтерфейсу користувача.

В загальному даний підхід до генерації бази та роботу з нею показав гарну роботу. Але неможливість використовувати дане API у вигляді REST запитів чи чогось більш інноваційного робить його достатньо специфічним, а це призведе до невеликою популярності даного підходу.

3 РОЗРОБКА МІКРО СЕРВІСУ З ВИКОРИСТАННЯМ NEO4J І GRAPHQL

Реляційні бази даних були потужними програмними додатками з 80-х років і залишаються зараз. Вони зберігають високо структуровані дані в таблицях з зумовленими стовпцями певних типів і безліччю рядків одного і того ж типу інформації і, частково завдяки жорсткості їх організації, вимагають від розробників додатків строго структурувати дані [26].

У реляційних базах даних посилання на інші рядки і таблиці вказуються шляхом посилання на їх (первинні) атрибути ключів через стовпці зовнішнього ключа. Це можна виконати з обмеженнями, але тільки тоді, коли посилання ніколи не є обов'язковим. Вибірка обчислюється під час запиту, зіставляючи первинні і зовнішні ключі багатьох рядків таблиць, які підлягають об'єднанню. Ці операції розраховані на обчислення і пам'ять і мають експонентну вартість [26].

Якщо ви використовуєте зв’язки «багато-до-багатьох», вам необхідно представити таблицю JOIN (або таблицю об’єднань), яка містить зовнішні ключі обох таблиць-учасників, що ще більше збільшує витрати на операції об'єднання. Ці дорогі операції об'єднання зазвичай вирішуються шляхом денормалізації даних, щоб зменшити кількість необхідних вибірок [26].

Хоча не кожен варіант використання підходить для такого типу точних моделей даних, в минулому відсутність життєздатних альтернатив і велика підтримка реляційних баз, даних ускладнювали створення альтернатив[26].

3.1 Від реляційних баз даних до графових

Зв’язки першокласних мереж у моделі графових баз даних, на відміну від інших систем управління базами даних, які вимагають від нас встановлення зв’язків між об'єктами, використовують спеціальні властивості, такі як зовнішні ключі. Об'єднавши прості абстракції вузлів і зв'язків в пов'язаних структурах, графові бази даних дозволяють нам створювати складні моделі, які тісно

пов'язані з нашою проблемною областю [26].

У деяких відносинах графові бази даних схожі на наступне покоління реляційних баз даних, але з підтримкою першого класу для «зв’язків» або з неявними сполуками, зазначеними за допомогою зовнішніх ключів в традиційних реляційних базах даних [26].

Кожен вузол (суб'єкт або атрибут) в моделі графових баз даних безпосередньо і фізично містить список взаємопов'язаних записів, які представляють його зв'язок з іншими вузлами. Ці взаємопов’язані записи організовані за типом і напрямком, і можуть містити додаткові атрибути. Всякий раз, коли ви запускаєте еквівалент операції JOIN, база даних просто використовує цей список і має безпосередній доступ до пов'язаних вузлів, що усуває необхідність в дорогому обчисленні пошуку / зіставлення [26].

3.1.1 Що таке графова база даних?

Графова база даних - це база даних, призначена для обробки зв’язків між даними як першокласна мережа в моделі даних. Ми живемо в взаємозв’язаному світі. В ньому немає ізольованих частин інформації, але багато, пов’язаних структур навколо нас. Лише база даних, яка спочатку підтримує зв’язки, здатна ефективно зберігати, обробляти і запитувати з'єднання. У той час як інші бази даних обчислюють їх під час запиту через дорогі операції JOIN[26].

Доступ до вузлів і зв’язків у графовій базі даних - це ефективна неперервна операція, яка дозволяє швидко зчитати мільйони підключень в секунду на вузол.

Незалежно від загального розміру вашого набору даних, графові бази даних перевершують управління високоповя’заними даними і складними запитами. Маючи тільки шаблон і набір вихідних точок, графові бази даних досліджують зв’язки навколо початкових точок - збір та його узагальнення інформації з мільйонів вузлів і зв’язків - залишаючи мільярди поза периметром пошуку недоторканими [26].

3.1.2 Модель властивостей графа

Якщо ви коли-небудь працювали з об'єктною моделлю або діаграмою зв’язків сутностей, модель з мітками властивостей буде здаватися вам знайомої.

Вузли - це об'єкти на графіку. Вони можуть містити будь-яку кількість атрибутів (пари ключ-значення), так звані властивості. Вузли можуть бути позначені ярликами, які представляють їхні різні ролі. На додаток до контекстуалізації властивостей вузла і зв’язків мітки можуть також служити для прикріплення інформації про індекс метаданих або обмеження до певних вузлів[26].

Зв’язки забезпечують направлення, іменовані, семантично релевантні зв'язку між двома вузловими об'єктами (наприклад, Employee WORKS_FOR Company). Зв’язок завжди має напрямок, тип, початковий вузол і кінцевий вузол. Як і вузли, зв’язки також можуть мати властивості. У більшості випадків відносини мають кількісні характеристики, такі як ваги, витрати, відстані, рейтинги, інтервали часу або стійкість. Оскільки відносини зберігаються ефективно, два вузла можуть ділитися будь-яким числом або типом зв’язків, не жертвуючи продуктивністю. Зверніть увагу, що, хоча вони спрямовані, зв’язки завжди можуть ефективно переміщатися в будь-якому напрямку[26].

3.1.3 Будівельні блоки властивостей графа

У базі даних графа є одне основне узгоджене правило: «Жодних зламаних зв’язків». Оскільки зв’язки завжди мають початковий і кінцевий вузол, ви не можете видалити вузол, не видаляючи також пов'язані з ним зв’язки. Ви також можете завжди припускати, що існуюче відношення ніколи не вкаже на неіснуючу кінцеву точку[26].

3.2 Опис можливостей Neo4j

Neo4j є відкритою базою даних NoSQL з відкритим вихідним кодом, яка забезпечує сумісний з ACID транзакційний сервер для ваших додатків. Розробки

ведуться з 2003 року, він був загальнодоступним з 2007 року. Вихідний код, написаний на Java і Scala, доступний на GitHub [26].

Neo4j використовується сьогодні тисячами компаній і організацій практично у всіх галузях, включаючи фінансові послуги, уряд, енергетику, технології, роздрібну торгівлю і виробництво. Сотні розробників і архітекторів в цих галузях - це сертифіковані професіонали Neo4j [26].

Деякі особливості роблять Neo4j дуже популярним серед розробників, архітекторів і адміністраторів баз даних[26]:

- Cypher, мова декларативних запитів, аналогічний SQL, але оптимізована для графів. Тепер використовується іншими базами даних, такими як графік SAP HANA Graph і Redis через проект openCypher.
- Дозволяє масштабувати до мільярдів вузлів на середньому апаратному забезпеченні.
- Гнучка схема властивостей графів, яка може адаптуватися з плином часу, дозволяючи згодом матеріалізувати і використовувати нові зв’язки для «швидкого доступу» до вузлів
- Драйвери для популярних мов програмування, включаючи Java, JavaScript, .NET і Python.

3.3 Реалізація на мові Cypher

Описання структур для графу буде проводитися на мові Cypher (https://neo4j.com/developer/cypher/).

Задача, яку буде вирішено: збереження медичних даних про пацієнта (особисті дані та результатів аналізів).

На Рис. 3.1 наведено код на мові Cypher, який необхідний для створення записів про пацієнта в графовій базі даних.

```
CREATE (Patient1:PATIENT_INSTANCE { was_born:'1900', has_location:'Kyiv', has_name:'Dima'})
CREATE (PatientData:PATIENT_DATA { has_timestamp: '2018-02-28-12:00'})

CREATE (Patient_data_1_01:PATIENT_DATA_ELEMENT { has_value: '80', of_type: 'BLOOD_PRESSURE_DIASTOLIC'})
CREATE (Patient_data_1_02:PATIENT_DATA_ELEMENT { has_value: '120', of_type: 'BLOOD_PRESSURE_SISTOLIC'})

CREATE
    (PatientData)-[:HAS_ELEMENT]->(Patient_data_1_01),
    (PatientData)-[:HAS_ELEMENT]->(Patient_data_1_02),
    (PatientData)-[:CREATED_BY]->(Patient1);
```

Рис. 3.1- Створення сутності Пацієнт

Напишемо вибірку, де вказуємо, які саме ті ноди, які ми хочемо отримати.

```
MATCH (e:PATIENT_DATA_ELEMENT)<-[:HAS_ELEMENT]-(d)-[CREATED_BY]->(p:PATIENT_INSTANCE)
RETURN d, e, p;
```

Рис. 3.2 – Запит на одержання графу

Результуючий граф зображено на Рис. 3.3. За допомогою графа можна прослідкувати всі зв’язки між вузлами та їх напрямки.

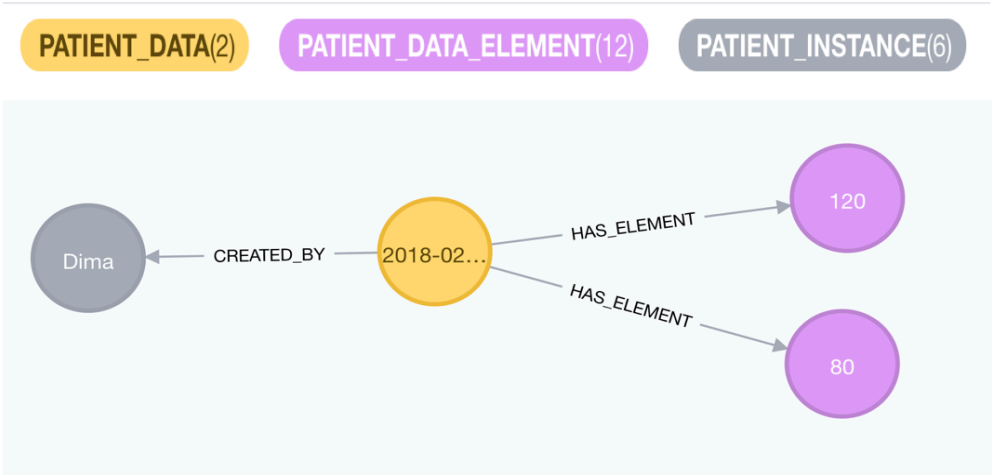


Рис. 3.3 – Результуючий граф

3.4 Рішення з використанням graphql

На початку 2018 року компанія Neo4j випустила розширення під назвою neo4j-graphql (https://github.com/neo4j-graphql/neo4j-graphql), яке дозволило виконувати запити в графову базу даних за допомогою GraphQL (https://en.wikipedia.org/wiki/GraphQL)[26].

Далі наведено повний перелік його функцій:

- Автогенерування всіх типів запитів для кожного елементу згідно до схеми
- Кожна властивість елементу доступна, як query аргумент
- Надано можливість сортувати результати при їх отриманні
- Також є вбудована курсорна пагінація
- Наявна валідація даних відповідно до схеми

Для початку роботи потрібно завантажити середовище для роботи за графовою базою даних під назвою neo4j-desktop, завантажити можна за посиланням (https://neo4j.com/download/). Після установки потрібно авторизуватись: логін – neo4j, пароль – admin. Вигляд середовища зображено на Рис.3.4

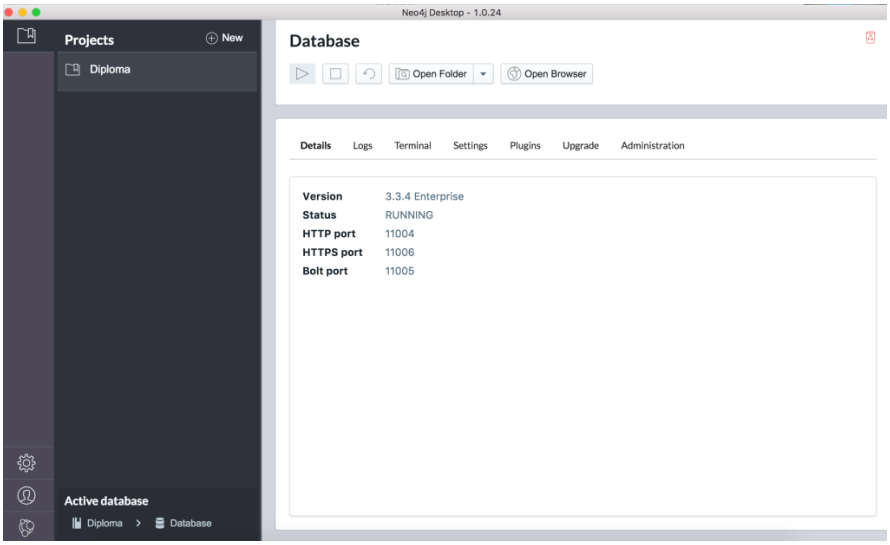


Рис. 3.4 - Загальний вигляд Neo4j Desktop

Наступним кроком створюємо базу та встановлюємо для неї плагін GraphQL. Він буде виступати провідною ланкою між нашими на GraphQL та GrapgDB, а якщо точніше, то він просто буде перетворювати запити з GraphQL на Cypher. Вікно установки показано на Рис. 3.5.

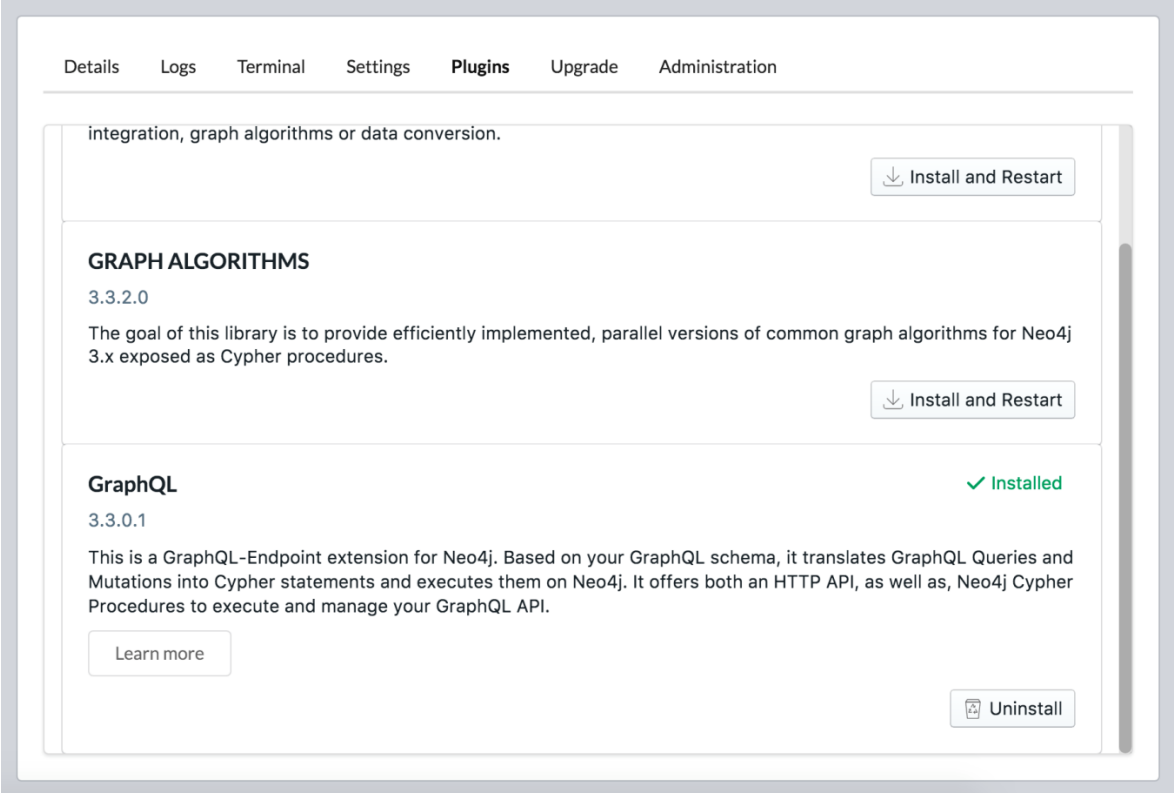


Рис.3.5–Вікно встановлення плагінів

3.4.1 Схему даних

Опишемо схему бази, використовуючи типи GraphQL. Базу поділено на 4 сутності: Doctor, Patient_Instance, Patient_Data, Patient_Data_Element.

- Лікаря (Doctor) буде описано за допомогою наступних полів: ідентифікатора, імені та пацієнтів. Де поле пацієнти – це масив сутностей.
- Пацієнта (Patient_Instance) - ідентифікатора, імені, часу додання в базу, дати народження, аналізів. Де поле аналізів - це масив сутностей.
- Аналіз (Patient_Data) – ідентифікатора, дати отримання та елементів цього аналізу, що теж може бути масивом. Наприклад: покази артеріального тиску мають два значення(верхній та нижній).
- Елемент аналізу – ідентифікатора, значення та типу цього аналізу.
- Типи аналізів це Enum: верхній тиск, нижній тиск, рівень цукру в крові, покази спірометра

myschema.graphql

```
type Doctor {
  id: ID
  name: String
  patients: [Patient_Instance]
}

type Patient_Instance {
  id: ID
  name: String
  timestamp: String
  born: String
  clinic: String
  data: [Patient_Data] @relation(name: "Created_By", direction: "IN")
}

type Patient_Data {
  id: ID
  created: String
  elements: [Patient_Data_Element]
    @relation(name: "Has_Element", direction: "IN")
}

type Patient_Data_Element {
  id: ID
  value: String
  type: [Patient_Data_Element_Type]
}

enum Patient_Data_Element_Type {
  BP_DIASTOLIC
  BP_SISTOLIC
  BS_LEVEL
  VT_CAPACITY
}
```

Рис. 3.6- GraphQL типи для нашої бази даних

Згенеруймо графову базу за допомогою команди:

CALL graphql.idl('schema-text')

Далі отримаємо базову схему в вигляді графа, запустивши команду:

`callgraphql.schema()`.

Результат показано на Рис. 3.7

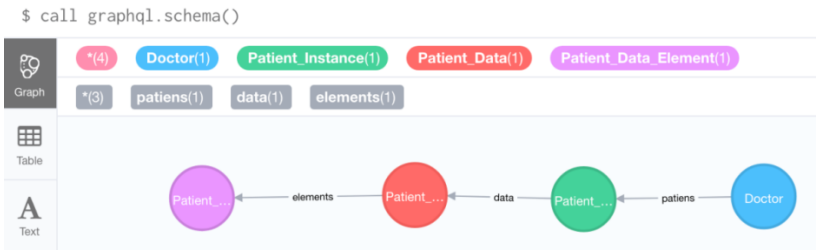


Рис. 3.7 – Загальний вигляд зв’язків між сутностями

3.4.2 Мутації

Для наповнення нашої бази даних даними будемо використовувати mutation об’єкти. Мутаціями в GraphQL називають будь-яку операцію, яка веде за собою змінення даних в базі. То можуть буди запити на створення, оновлення чи видалення елементів.

Для виклику мутацій будемо використовувати середовище GraphiQL. Наше GraphQL API буде доступне за адресою http://localhost:7474/graphql/. Перш ніж його використовувати потрібно згенерувати авторизаційний токен:

BasicAuthorization: bmVvNGo6cG9zaXRpb24tYm9uZC1tYW5ldXZlcg=,

та помістити його в хедер. Токен має бути згеровано за допомогою програми, яка зображена на рис.3.8 з використання вашого логіну та паролю то neo4j-desktop.

Рис. 3.8 – Утіліта для конвертації логіну та паролю в base64 токен

Тепер можемо перейти до мутацій. Так, наприклад, для створення пацієнта з набором аналізів: два заміри артеріального тиску з різницею в часі, замір рівня цукру та показ спірометра буде виглядати наступним чином.

GraphQL Endpoint GraphQL Endpoint

GraphiQL ▶ Prettify History

```
mutation {
    d1: createDoctor(id: "1", name: "Doctor House")

    p1: createPatient_Instance(id:"1", name:"Karl", timestamp: "1994", born: "New York")

    data1_1: createPatient_Data(id: "1_1", created: "01-04-2018" )
    data1_2: createPatient_Data(id: "1_2", created: "02-03-2018")
    data1_3: createPatient_Data(id: "1_3", created: "03-02-2018")
    data1_4: createPatient_Data(id: "1_4", created: "04-01-2018")

    element1_1_1: createPatient_Data_Element(id: "111", value: "120", type: BP_SISTOLIC)
    element1_1_2: createPatient_Data_Element(id: "112", value: "80", type: BP_DIASTOLIC)
    element1_2: createPatient_Data_Element(id: "12", value: "4.5", type: BS_LEVEL)
    element1_3: createPatient_Data_Element(id: "13", value: "4.7", type: BS_LEVEL)
    element1_4: createPatient_Data_Element(id: "14", value: "2300", type: VT_CAPACITY)

    link1: addPatient_InstanceData(id:"1", data:["1_1", "1_2", "1_3", "1_4"])

    cast1_1: addPatient_DataElements(id: "1_1", elements:["111", "112"])
    cast1_2: addPatient_DataElements(id: "1_2", elements:["12"])
    cast1_3: addPatient_DataElements(id: "1_3", elements:["13"])
    cast1_4: addPatient_DataElements(id: "1_3", elements:["14"])

    doc: addDoctorPatiens(id: "1", patiens: ["1"])
}
```

Рис. 3.9 - Мутація на створення пацієнта з аналізами. Виконується в середовищі GraphiQL

Результатом виконання подібної мутації тільки для трьох пацієнтів буде наступний граф.

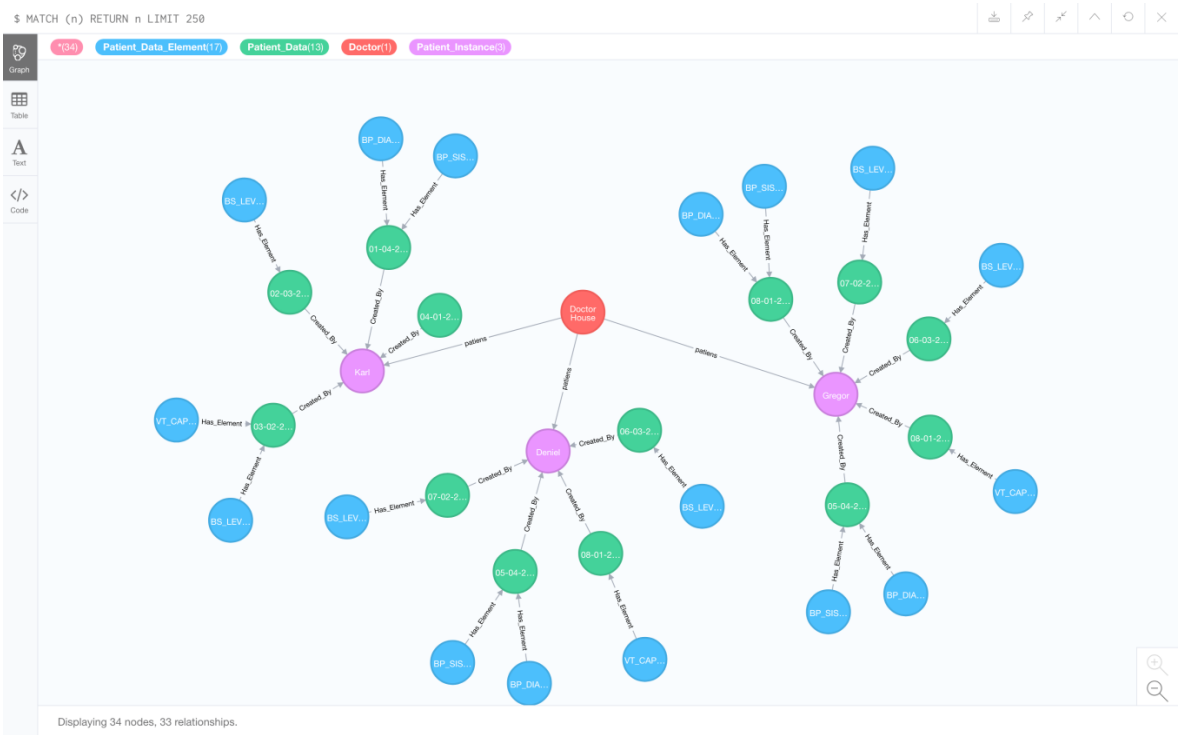


Рис. 3.10 - Загальний вигляд структури графової бази даних у інтерфейсі Neo4j Desktop

А якщо збільшити кількість пацієнтів до 8, то результуючий граф буде мати вже зовсім інший вигляд. Всіх пацієнтів було приписано до одного лікаря. Але зрозуміло, що структура бази буде розростатися, якщо додати інших лікарів та клініки.

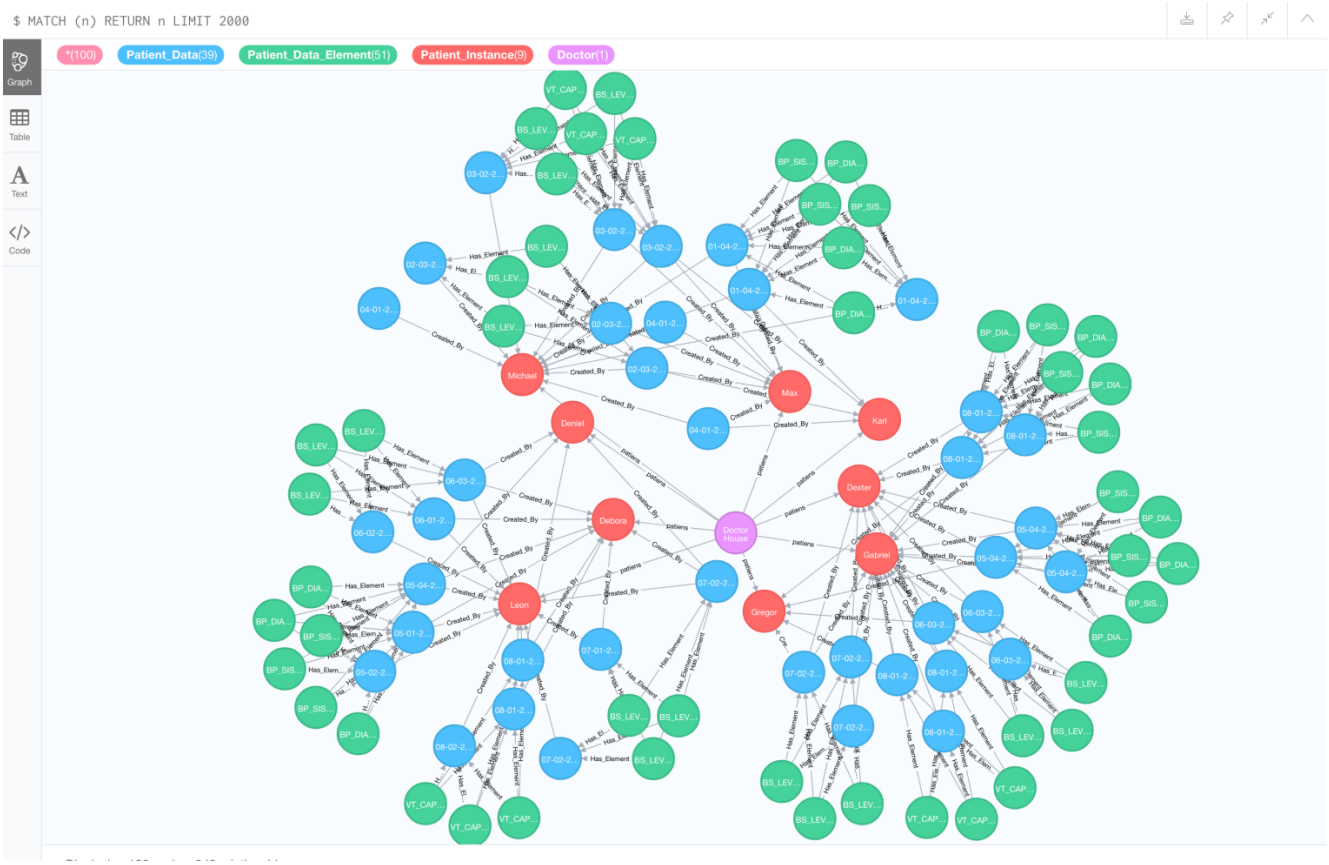


Рис. 3.11 – Вигляд графу з восьми пов’язаних сутностей

Повний список доступних мутацій нашої бази доступний в інтерфейсі як GraphiQL так, і як у вигляді готових роутів, що можна викликати з сайту для роботи з нашим сервісом зображено на Рис. 3.12

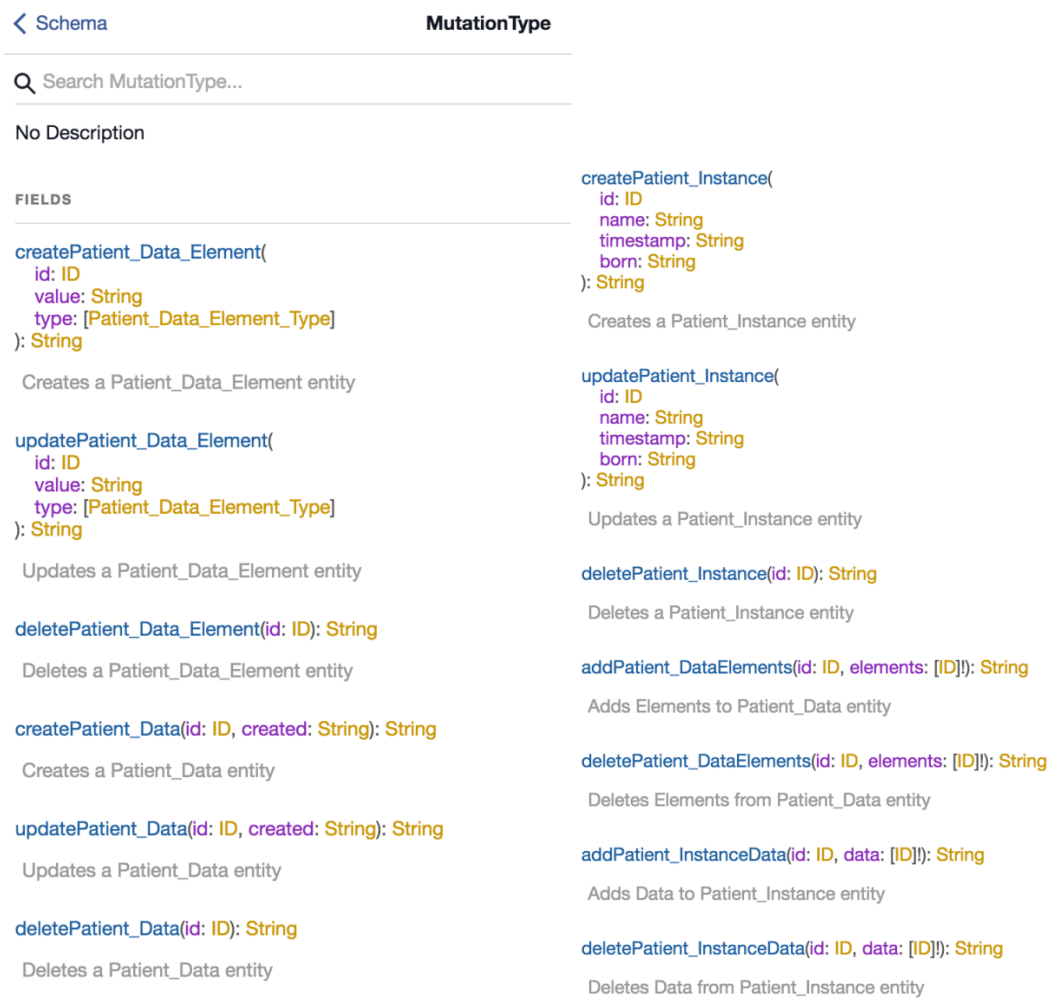


Рис. 3.12 – Повний список доступних мутацій

3.4.3 Отримання даних за допомогою сервісу

Для отримання даних за допомогою сервісу використовуються не мутації, а query.

Для прикладу, для того щоб отримати об’єкт першого пацієнта з полями, які нам потрібні, достатньо зробити лише один виклик API. Текст запиту(зліва) та відповіді API(зправа) зображено на рис. 3.13.

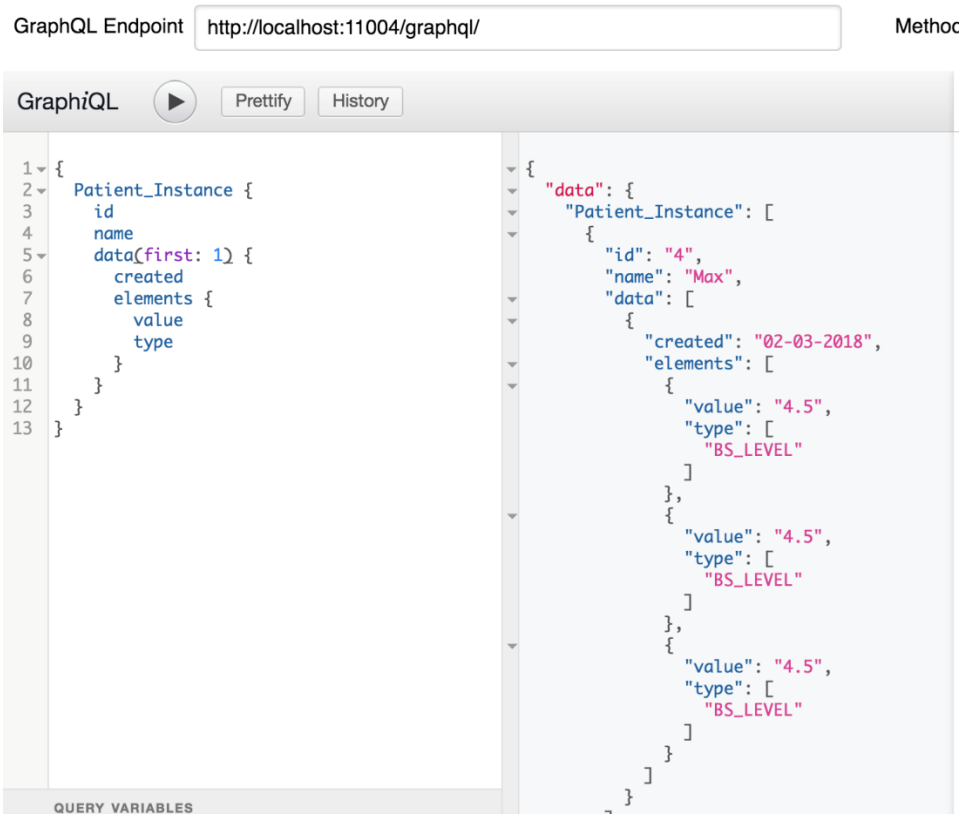


Рис. 3.13 – Приклад отримання даних у форматі JSON за допомогою GraphQLзапиту

Повний список параметрів для отримання даних з бази містить в собі не тільки самі значення полів в об’єктах, а і найрізноманітніші варіанти сортування та вибірок цих даних. Всі ці параметри можна переглянути в GraphiQL панелі рис. 3.14.

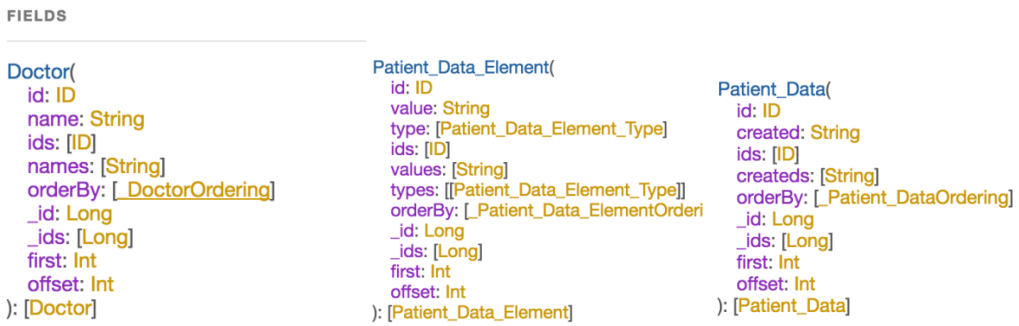


Рис. 3.14 – Розгорнутий вигляд сутностей та полів, що доступні

3.4.4 Процес отримання Docker Image

Процес починається з отримання базового образу для нашої GraphDB бази. Спочатку потрібно написати Dockerfile, в якому прописати необхідні плагіни, які необхідно додати до нашого майбутнього образу.

```
FROM neo4j
ARG NEO4J_GRAPHQL_RELEASE
EXPOSE 7474
RUN wget -P /var/lib/neo4j/plugins $NEO4J_GRAPHQL_RELEASE
RUN echo 'dbms.unmanaged_extension_classes=org.neo4j.graphql=/graphql' >>
/var/lib/neo4j/conf/neo4j.conf
```

Далі потрібно зібрати контейнер, використовуючи наступну команду, яка зображена на Рис. 3.15

```
~/Desktop/neo4j-graphql-docker master*
❯ docker build -t neo4j-graphql --build-arg NEO4J_GRAPHQL_RELEASE='https://github.com/neo4j-graphql/neo4j-graphql/releases/download/3.3.0.1/neo4j-graphql-3.3.0.1.jar' .
Sending build context to Docker daemon  2.016MB
Step 1/5 : FROM neo4j
 ---> a41ed0035eee
Step 2/5 : ARG NEO4J_GRAPHQL_RELEASE
 ---> Using cache
 ---> caf2031fea4d
Step 3/5 : EXPOSE 7474
 ---> Using cache
 ---> ec8ede7a0e1d
Step 4/5 : RUN wget -P /var/lib/neo4j/plugins $NEO4J_GRAPHQL_RELEASE
 ---> Using cache
 ---> a1652bdf3a58
Step 5/5 : RUN echo 'dbms.unmanaged_extension_classes=org.neo4j.graphql=/graphql' >> /var/lib/neo4j/conf/neo4j.conf
 ---> Using cache
 ---> 2e4f2cadc2d9
Successfully built 2e4f2cadc2d9
Successfully tagged neo4j-graphql:latest
```

Рис. 3.15 – Процес збирання контейнера

Після того, як контейнер готовий, можемо спробувати запустити його на локальній машині. Процес запуску зображено на Рис. 3.16.

```
~/Desktop/neo4j-graphql-docker master*
❯ docker run -p 7474:7474 -p 7687:7687 neo4j-graphql
Active database: graph.db
Directories in use:
  home:         /var/lib/neo4j
  config:       /var/lib/neo4j/conf
  logs:         /var/lib/neo4j/logs
  plugins:      /var/lib/neo4j/plugins
  import:       /var/lib/neo4j/import
  data:         /var/lib/neo4j/data
  certificates: /var/lib/neo4j/certificates
  run:          /var/lib/neo4j/run
Starting Neo4j.
```

Рис. 3.16 – Процес запуску контейнера на локальній машині

Тепер відкриваємо порт localhost:7474 та можемо повноцінно використовувати наш мікросервіс для роботи за базою.

3.5 Висновки

В даному розділі було розроблено повноцінний мікросервіс на базі GraphQL API, що дає можливість працювати за графовою базою даних. Записувати до графової бази даних ми можемо тільки відповідно до семантичної схеми бази, яка була описана, використовуючи типи grapgql. Всі ноди отримали певні властивості та зв'язки з іншими нодами. Кожна властивість має свій тип, що унеможливлює запис до бази невалідних об'єктів, а також унеможливлює зв'язання несумісних сутностей.

Записувати дані до бази можна, використовуючи GraphQL мутації, при цьому у разі не консистентних даних ми отримаємо помилку. Переглянути всю базу ми можемо завдяки Neo4j Desktop. Цей додаток має дуже зручний інтерфейс роботи з графовими базами даних. Було надано список всіх доступних мутацій, що ми можемо виконувати з нашими даними.

Дані можна отримувати за допомогою GraphQL запитів, при цьому очевидна перевага над REST API. Адже наша технологія дає можливість

вказувати, які саме сутності ми хочемо отримати, які поля нам потрібні для роботи, та які їх зовнішні зв’язки нам потрібно. Так, щоб отримати структуру будь-якої складності, нам достатньо здійснити всього лише один запит. При цьому є можливість кешувати кожен запит, або не робити повторні запити на отримання даних. Дані отримуються в форматі JSON, що зручно, адже це найпопулярніший формат, на відміну від застарілого XML.

Мікросервіс було зібрано в Docker контейнер та розміщено на DockerHub.

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ “РЕЄСТР ВЕБ СЕРВІСІВ З СЕМАНТИЧНИМИ АНОТАЦІЯМИ ДЛЯ МОБІЛЬНОЇ МЕДИЧНОЇ ПЛАТФОРМИ”

Розділ має на меті проведення маркетингового аналізу стартапу проекту “Реєстр веб-сервісів з семантичними анотаціями для мобільної медичної платформи” задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження.

Метою розділу є формування інноваційного мислення, підприємницького духу та формування здатностей щодо оцінювання ринкових перспектив і можливостей комерціалізації основних науково-технічних розробок, сформованих у попередній частині магістерської дисертації у вигляді розроблення концепції стартап-проекту “Реєстр веб-сервісів з семантичними анотаціями для мобільної медичної платформи” в умовах висококонкурентної ринкової економіки глобалізаційних процесів.

Опис стартап-проекту “База знань як сервіс” наведено у Таблиці 4.1.

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Створення веб сервісу для роботи з даними пацієнтів з наданням користувачеві доступу читання та запису через GraphQLapi. Надання GraphQL-ендпойнту для читання та оновлення даних у графовій базі даних.	1. Використання для виконання аналізу даних про пацієнтів, що зберігаються у вигляді графу.	GraphQL API надає можливість використовувати систему як сервіс, що значно спрощує доступ до сховища. Знаходження транзитивних відношень, класів та об’єктів онтологій може спростити подальший аналіз даних.

	2. Використання сервісу як одного з набору сервісів системі та можливість їх інтеграції	Можливість виконання GraphQl запитів дає можливість гнучкого доступу до даних та їх оновлення. Сервіс легко інтегрувати у інші системи завдяки використанню Gateway

Отже, проект “Реєстр веб-сервісів з семантичними анотаціями для мобільної медичної платформи” може бути використаним як інструмент для деякого аналізу даних, так і прошарок постійного збереження сильно пов’язаних доменних даних у інфраструктурі інших додатків завдяки можливості використання даної системи як сервісу через GraphQL API.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	*Техніко-економічні характерис-тики ідеї*	*(потенційні) товари/концепції конкурентів*				*W (слабка сторона)*	*N (нейтра- льна сторона)*	*S (сильна сторона)*
		Мій проект	*Конкурент 1*	*Конкурент2*	*Конкурент 3*			
1.	Форма виконання	Веб-сервіс	Програма	Веб-додаток	Програма			+
2.	Собівартість	Низька	Низька	Висока	Висока			+
3.	Кросплатформність	Так	Ні	Так	Так			+

4.	Наявність GraphQl-ендпойнту для зчитування даних	Так	Так	Так	Так		+	
5.	Наявність GraphQl-ендпойнту для оновлення даних	Так	Ні	Ні	Так			+
6.	Застосування логічного виведення	Так	Ні	Так	Так			+
7.	Горизонтальне масштабування	Так	Так	Ні	Так			+

Сильними сторонами проекту є форма виконання у вигляді веб-сервісу, новий підхід у побудові мікросервісу, низька собівартість, кросплатформність, наявність горизонтального масштабування. Наявність GraphQL-ендпойнту для оновлення даних.

4.1 Технологічний аудит ідеї проекту

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару).

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Створення графової бази даних	Neo4j	Наявна	Безкоштовна, доступна

		GraphDB	Наявна	Частково безкоштовна
2.	Створення GraphQl API для доступу до бази даних	Neo4j-GraphQl	Наявні	Безкоштовна, доступна
			Наявна	Платні
3.	Розгортання контейнера	DockerHub	Наявна	Безкоштовна, доступна
		Amazon	Наявна	Платні

Обрані технології реалізації ідеї проекту: Neo4j через повну безкоштовність фреймворку та наявність докладної документації, наявність досвіду роботи розробників з даною технологією; Neo4j-GraphQl через простоту використання, безкоштовність та можливість розгортання додатків на основі таких технологій у хмарі; DockerHub для розгортання у хмарі через безкоштовність.

4.2 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1.	Кількість головних гравців, од	5
2.	Загальний обсяг продаж, грн/ум.од	8000 грн./ум.од
3.	Динаміка ринку (якісна оцінка)	Зростає
4.	Наявність обмежень для входу (вказати характер обмежень)	Немає
5.	Специфічні вимоги до стандартизації та сертифікації	Немає
6.	Середня норма рентабельності в галузі (або по ринку), %	R = (3000000 * 100) / (1000000 * 12) = 25%

Отже, було проаналізовано наявність попиту, обсяг, динаміку розвитку ринку. Обмеження для входу на ринок відсутні, динаміка ринку зростає, галузь є рентабельною.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи (табл. 5).

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп*	*Вимоги споживачів до товару*

			клієнтів	
1.	Необхідне програмне забезпечення(GraphQL API) для доступу до бази	Потенційними цільовими групами є медичні заклади які хочуть отрати гнучку систему для роботи з даними пацієнтів	немає	Рішення повинне бути придатним до інтеграції в інші більш складні системи, мати GraphQl-endpoint, бути здатним надавати дані в тому вигляжі якому їх хочуть отримати, розгорнутим на DockerHub

Згідно проведеної характеристики потенційних клієнтів стартап-проекту випливає, що на ринку є затребуваним програмне забезпечення (GraphQl API) для доступу до графової бази даних як до мікросервісу.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. № 6-7). Фактори в таблиці подавати в порядку зменшення значущості.

Таблиця 4.6 – Фактори загроз

№ п/п	*Фактор*	*Змістзагрози*	*Можлива реакція компанії*
1.	Конкуренція	Вихід на ринок великої компанії	1. Вихід з ринку 2. Запропонувати великій компанії поглинути себе 3. Передбачити додаткові переваги власного ПЗ для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок
2.	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	1. Передбачити можливість додавання нового функціоналу до створюваного ПЗ
3.	Зміна тарифів провайдера хмарного розгортання на платні	Необхідність оплати послуг провайдера хмари	1. Пошук іншого безкоштовного провайдера 2. Пошук інвестицій для оплати існуючого провайдера
4.	Надходження на ринок альтернативних продуктів	Перехід користувачів нашого товару на інший продукт	Впровадження нового функціоналу, якого немає у конкурентів

5.	Уповільнення росту ринку	Скорочення користувачів продуктів, що тільки виходять на ринок	Інвестиції у впровадження ефективної реклами продукту

Отже, було проаналізовано фактори загроз ринкового впровадження проекту, серед яких: конкуренція, уповільнення росту ринку, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні та надходження на ринок альтернативних продуктів. Було також запропоновано можливі реакції компанії.

Таблиця 4.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1.	Стрімкий ріст попиту на інструменти обробки даних у вигляді графових баз даних	Наявність попиту на інструменти для обробки даних у вигляді о графових баз даних	Змога запропонувати продукт більшій кількості потенційних користувачів
2.	Стрімке зростання росту ринку	Компаніям, що тільки виходять на ринок, буде простіше отримати клієнтів	Змога запропонувати продукт більшій кількості потенційних користувачів
3.	Обслуговування додаткових груп споживачів	Поява нових потенційних груп споживачів	Змога розширити продукт для подальшого впровадження у нові галузі

4.	Розширення асортименту можливих послуг	Поява нового функціоналу, що привабить нових користувачів	Розробка нового функціоналу, що є потребою певної групи користувачів

У Таблиці 4.7 наведено фактори можливостей ринкового впровадження проекту, серед яких: стрімкий ріст попиту на інструменти обробки даних у вигляді графових баз даних, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг; було також запропоновано можливі реакції компанії.

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (табл. 8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції - досконала	Існує 3 фірми- конкурентки на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2. За рівнем конкурентної боротьби - міжнародний	Одна з компаній – з іншої країни, дві – з України	Додати можливість вибору мови ПЗ, щоб легше було у майбутньому вийти на міжнародний ринок

3. За галузевою ознакою - внутрішньо галузева	Конкуренти мають ПЗ, яке використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях
4. Конкуренція за видами товарів: - товарно-видова	Види товарів є однаковими, а саме – програмне забезпечення	Створити ПЗ, враховуючи недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж використовують конкуренти
6. За інтенсивністю - не марочна	Бренди відсутні	-

У Таблиці 4.8 наведено ступеневий аналіз конкуренції на ринку, де було визначено особливості конкурентного середовища та їх вплив а діяльність підприємства. Однією з найбільш важливих дій компанії для досягнення конкурентоспроможності є необхідність створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (Табл. 9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складов і аналізу	*Прямі конкуренти в галузі*	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*

	Навести перелік прямих конкурентів	*Визначити бар'єри входження в ринок*	*Визначити фактори сили постачальників*	*Визначити фактори сили споживачі в*	*Фактори загроз з боку замінників*
Висновки:	Існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 2, так як його рішення також представлене у вигляді веб-додатку	Так, можливості для входу на ринок є, бо наше рішення має GraphQL-ендпойнт для оновлення даних та можливість логічного виведення	Постачальники відсутні.	Важливим для користувача є швидкість роботи ПЗ	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару

Було здійснено аналіз конкуренції в галузі за М. Портером, в результаті чого було визначено, що існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 2, так як його рішення також представлене у вигляді веб-додатку, але можливості для входу на ринок є, бо наше рішення має GraphQL-ендпойнт для оновлення даних та можливість логічного виведення.

За результатами аналізу таблиці робиться висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також робиться висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспроможності у п. 3.6. 3.6) На

основі аналізу конкуренції, проведеного в п. 3.5 (табл. 9), а також із урахуванням характеристик ідеї проекту (табл. 2), вимог споживачів до товару (табл. 5) та факторів маркетингового середовища (табл. № 6-7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за табл. 10

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1.	Наявність GraphQL-endpoint для зчитування та оновлення даних	Дозволяє користувачам здійснювати гнучкий доступ до бази знань
2.	Можливість завантажувати цілі онтології	Висока швидкість заповнення бази знань
3.	Наявність GraphQL API	Дозволяє інтегрувати сервіс у складні системи завдяки універсальному API
4.	Хмарне розгортання	Дозволяє звертатись до бази знань як до сервісу
5.	Горизонтальне масштабування	Можливість гнучкого масштабування за допомогою додавання апаратних компонентів

У Таблиці 7 наведено обґрунтування факторів конкурентоспроможності, серед яких: наявність GraphQL-endpoint для зчитування та оновлення даних,

можливість завантажувати цілі онтології, наявність GraphQL API та хмарне розгортання. Було також наведено обґрунтування цих факторів.

За визначеними факторами конкурентоспроможності (табл. 10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 11).

У наступній таблиці наведено проведення аналізу сильних та слабких сторін стартап-проекту, факторами конкурентоспроможності виступили такі: наявність GraphQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність REST API, хмарне розгортання, горизонтальне масштабування.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з нашим підприємством*						
			-3	-2	-1	0	1	2	3
1.	Наявність GraphQL-endpoint для зчитування та оновлення даних	20				+			
2.	Можливість завантажувати цілі онтології	20			+				
3.	Наявність GraphQL API	15			+				
4.	Хмарне розгортання	15		+					
5.	Горизонтальне масштабування	10					+		

Отже, серед сильних сторін проекту можна виділити наступні: наявність GraphQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність GraphQL API, можливість хмарного розгортання.

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити 103 прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

У наступній таблиці буде проілюстровано SWOT-аналіз стартап-проекту, тобто його слабкі та сильні сторони, можливості та загрози виходу на ринок.

Таблиця 12. SWOT- аналіз стартап-проекту

Сильні сторони: наявність GraphQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність GraphQL API, можливість хмарного розгортання	Слабкі сторони: можливість зміни тарифів провайдером хмарного розгортання на платні, відсутність можливості горизонтального масштабування
Можливості: стрімкий ріст попиту на інструменти обробки даних у вигляді графових баз даних, стрімке зростання росту ринку, обслуговування додаткових груп споживачів,розширення асортименту можливих послуг	Загрози: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Створення додатку з використанням фреймворків ApacheJena, SpringBoot	60%	6 місяців
2.	Створення програми на основі без використання будь- яких фреймворків для обробки даних	35%	8 місяців

З означених альтернатив обирається та, для якої: а) отримання ресурсів є більш простим та ймовірним; б) строки реалізації – більш стислими. Тому обираємо альтернативу (створення додатку з використанням фреймворків ApacheJena, SpringBoot).

4.3 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 14).

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовни й попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*

1.	Дослідницькі центри	Спрощення роботи з високо пов’язаними даними	Великий	Існує 3 конкуренти, які надають схожі, але більш вузькі і дорогі рішення.	Наявність GraphQL API, GraphQL-ендпойнту, логічного виведення
2.	Клініки	Спрощення роботи з високо пов’язаними даними	Великий		Можливість інтеграції в уже існуючі системи завдяки REST API, зручне хмарне розгортання, наявність SPARQL-ендпойнту
Які цільові групи обрано: обираємо клініки та дослідницькі центри					

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку. Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку. За М. Портером, існують три базові стратегії розвитку, що відрізняються за ступенем охоплення цільового ринку та типом конкурентної переваги, що має бути реалізована на ринку (за витратами або визначними якостями товару).

Отже, проілюструвати базову стратегію розвитку можна у виляді Таблиці 4.15

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможніпозиції відповідно до обраної альтернативи*	*Базова стратегія розвитку*
1.	Створення веб-сервісу, використовуючи Ne04j, GrapgQl, GraphDB	Ринкове позиціонування	Можливість інтеграції в уже існуючі системи завдяки GraphQL API, зручне хмарне розгортання, наявність GraphQL-ендпойнту, логічного виведення	Диференціація

Було обрано таку альтернативу розвитку проекту: створення веб-сервісу, використовуючи Ne04j, GrapgQl, GraphDB адже завдяки цим технологіям можна досягнути ключових конкурентноспроможних позицій кінцевого продукту.

Таблиця 4.16 - Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки*

1.	Так	Так	Буде, а саме: основною задачею є розробка ПЗ з використанням сховища триплетів(конкуренти 1, 2, 3), форма виконання - веб-сервіс (конкурент 2)	Зайняттяконкурентної ніші

Отже, було визначено базову стратегію конкурентної поведінки як зайняття конкурентної ніші.

Визначимо стратегію позиціонування у Таблиці 4.17, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 - Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегі я розвитку*	*Ключові конкурентоспроможні позиції власного стартап- проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1.	Наявність універсального API, зручне хмарне розгортання, наявність GraphQL-ендпойнту, логічного виведення	Диференціація	Можливість інтеграції в уже існуючі системи завдяки GraphQL API, зручне хмарне розгортання, наявність GraphQL-ендпойнту, логічного виведення	Інтеграція, хмарне розгортання, GraphQL-ендпойнт

Отже, було вибрано такі асоціації, які мають сформувати комплексну позицію власного проекту: інтеграція (адже завдяки GraphQL API сервіс просто

інтегрувати у існуючі системи), хмарне розгортання, GraphQL-ендпойнт (у системі є повноцінний GraphQL-ендпойнт).

4.4 Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 4.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 - Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)*
1.	Наявність універсального API	Додаток реалізований у вигляді GraphQL сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів	Перевага в універсальності на можливості інтегрувати сервіс у існуючі системи.
2.	Можливість зручного хмарного розгортання	Можливість розгорнути додаток всюди, де є функціонал розгортання докер контейнера	Користувачі мають змогу працювати з системою віддалено у хмарі

3.	Наявність GraphQL-ендпойнту,	Можливість виконання будь-яких GraphQL-запитів	Підтримка стандарту GraphQL
4.	Наявність логічного виведення	Виведення транзитивних відношень у онтологіях	Можливість виведення транзитивних відношень у онтологіях

Отже бачимо, що проект має ключові переваги перед конкурентами, які повністю відповідають потребам цільової аудиторії. Додаток реалізований у вигляді GraphQL сервісу, що надає відповіді у вигляді JSON, що дає змогу користувачам звертатись до сервісу за допомогою стандартних HTTP POST та GET запитів, а це є досить універсальним способом для подальшої інтеграції сервісу в інші системи.

Далі у Таблиці 4.19 проілюстрована трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання.

Таблиця 4.19 - Опис трьох рівнів моделі товару

Рівні товару	*Сутність та складові*		
I. Товар за задумом	Веб-сервіс, що надає доступ до сховища триплетів за допомогою HTTP запитів, дозволяє працювати зі GraphQL-ендпойнтом та надає можливості логічного виведення та хмарного розгортання		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор

	1. Наявність універсального API 2. Можливість зручного хмарного розгортання 3. Наявність GraphQL-ендпойнту, 4. Наявність логічного виведення	1.Нм 2.Нм 3.Нм 4.Нм	1.Технологічна 2.Технологічна 3.Технологічна 4.Технологічна
	Якість: згідно до стандарту ISO 4444 буде проведено тестування		
	Маркування відсутнє		
	Моя компанія: “Graphfuture”		
III. Товар із підкріплюнням	1-місячна пробна безкоштовна версія		
	Постійна підтримка для користувачів		
За рахунок чого потенційний товар буде захищено від копіювання: патент			

Було описано три рівні моделі товару, з чого можна зробити висновок, що основні властивості товару у реальному виконанні є нематеріальними та технологічними. Також було надано сутність та складові товару у задумці та товару з підкріпленням.

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. У даному випадку найбільш вірогідним гарантом буде патент.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 4.20). Аналіз проводиться експертним методом.

Таблиця 4.20 - Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники, грн.*	*Рівень цін на товари-аналоги, грн.*	*Рівень доходів цільової групи споживачів, грн.*	*Верхня та нижня межі встановлення ціни на товар/послугу, грн.*
1.	45000	38000	150000	35000-40000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 4.21).

Таблиця 4.21 - Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Придбання підписки та оплата щомісячних внесків для продовження ліцензії	Продаж	0(напряму), 1(через одного посередника)	Власна та через посередників

Отже, система приноситиме прибуток завдяки щомісячним внескам для подовження ліцензії та придбанням підписок, продаж будк виконуватись напряму або через одного посередника.

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.22).

Таблиця 4.22 - Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуютьс я цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1.	Придбання ліцензії на користування в мережі Інтернет, щомісячне її продовження, користування сервісом у хмарі або ж на власних серверах.	Інтернет	Інтеграція, хмарне розгортання, GraphQL-ендпойнт	Показати переваги сервісу, у тому числі і перед конкурентами	Демо-ролик із використання, рекламні оголошення на популярних сайтах.

Отже, в Таблиці 4.22 наведено концепцію маркетингових комунікацій, було визначено, що придбання ліцензії на користування буде здійснюватись в мережі Інтернет, необхідним буде щомісячне її продовження, користування сервісом можливе у хмарі або ж на власних серверах.

4.5 Висновки

Згідно до проведених досліджень існує можливість ринкової комерціалізації проекту. Також, варто відмітити, що існують перспективи впровадження з огляду на потенційні групи клієнтів, бар‘єри входження не є високими, а проект має дві значні переваги перед конкурентами. Для успішного виконання проекту необхідно реалізувати програму із використанням засобів Neo4j, GraphQL, GraphDB. Для успішного виходу на ринок у продукту повинні бути наступні характеристики:

- наявність універсального API

- можливість зручного хмарного розгортання
- наявність GraphQL-ендпойнту
- наявність логічного виведення

В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

Було визначено такі сильні сторони: наявність GraphQL-endpoint для зчитування та оновлення даних, можливість завантажувати цілі онтології, наявність GraphQL API, можливість хмарного розгортання. Серед слабких сторін можна виділити можливість зміни тарифів провайдером хмарного розгортання на платні, відсутність можливості горизонтального масштабування.

Можливості для виходу на ринок включають стрімкий ріст попиту на інструменти обробки даних у вигляді графів, стрімке зростання росту ринку, обслуговування додаткових груп споживачів, розширення асортименту можливих послуг. Наявні такі фактори загроз: конкуренція, зміна потреб користувачів, зміна тарифів провайдера хмарного розгортання на платні, надходження на ринок альтернативних продуктів, уповільнення росту ринку.

Висновки

Було досліджено методи та інструменти для створення семантичного веб-сервісу. Розглянуто проблеми опису такого сервісу. Було сформовано необхідні умови для створення такого сервісу.

Для вирішення цього завдання було запропоновано два альтернативних рішення. Обидва були на основі Neo4j, адже це дуже відома у всьому світі платформа для роботи з графовими базами даних. Її вже використовують такі компанії, як Amazon, IBM, Cisco, Microsoft. Але в першому випадку ми дали можливість описати семантику сервісу за допомогою RDF/OWL як то було в 2005 році, коли альтернатив не було, або за допомогою синтаксису Cypher. А в другому випадку була використана така інноваційна технологія як GraphQL.

Основний внесок цієї роботи полягає в наступному. По-перше, описано інноваційний підхід до створення мікросервісу на основі Neo4jта GraphQL, який з’явився в 2018 році. Що дало нам велику перевагу у порівняні з подібними рішеннями на Java. Адже поріг для створення такого сервісу є набагато нижчим, а також це дає можливість легко підтримувати такі мікросервіси. Такий сервіс може бути зібрано в Docker контейнер, що знову ж таки спрощує інтеграцію з іншими не схожими на цей мікро сервісами. При цьому варто зауважити, що інтеграція таких мікро сервісів буде виконуватися на спільному Gateway.

По-друге, було продемонстровано роботу даного мікросервісу на етапах наповнення його графової бази даних даними, на етапі отримання даних з бази за допомогою GraphQL запитів у форматі JSON. Образ даного сервісу було поміщено на DockerHub.

Також в рамках даного дослідження була запропонована реалізація стартап-проекту, розраховані основні фінансово-економічні показники та проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що існують перспективи впровадження з

огляду на потенційні групи клієнтів, бар‘єри входження не є високими, а проект має дві значні переваги перед конкурентами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bernstein, C. Kiefer: Imprecise RDQL: Towards Generic Retrievalin Ontologies Using Similarity Joins. Proc. ACM Symposium on Applied Computing, Dijon, France, ACM Press, 2006.
2. D. Calvanese, G. DeGiacomo, I. Horrocks, C. Lutz, B. Motik, B. Parsia, P. Patel-Schneider: OWL 1.1 Web Ontology Language Tractable Fragments. W3C Member Submission, 19 December 2006. www.w3.org/Submission/2006/SUBM-owl11-tractable-20061219/. Updated version at www.webont.org/owl/1.1/tractable.html (6 April 2007)
3. J. Cardoso, A. Sheth (Eds.): Semantic WebServices: Processes and Applications. Springer book serieson SemanticWeb&Beyond: Computing for human Experience, 2006.
4. D. Connolly, F. Van Harmelen, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein: DAML+OIL referencedescription. W3C Note, 18 December 2001. Availableat www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218.
5. D. Fensel, H. Lausen, A. Polleres, J. deBruijn, M. Stollberg, D. Roman, J. Domingue: EnablingSemantic Web Services— The Web Service Modeling Ontology. Springer, 2006.
6. D. Fensel, F. Van Harmelen: Unifying reasoning and search to Webscale. IEEE InternetComputing, March/April 2007.
7. Glimm, I. Horrocks, C. Lutz, U. Sattler: Conjunctive Query Answering for the Description Logic SHIQ. Proceeding sof International Joint Conferenceon AI (IJCAI), 2007.
8. S. Grimm: Discovery— Identifying relevant services. In [24], 2007.

9. Grosof, I. Horrocks, R. Volz, S. Decker: Description Logic Programs: Combining Logic Programs with Description Logic. Proceedings of the 12th International World Wide Web Conference (WWW), 2003.
10. B. He, M. Patel, Z. Zhang, K.Chang: Accessingt he Deep Web. Communication sof the ACM, 50(5), 2007.
11. Horrocks, P. Patel-Schneider: Reducing OWL entailment to description logic satisfiability. (ISWC), 2003, Springer, LNCS, 2870, 2003.
12. Horrocks, P. Patel-Schneider: A proposal for an OWL rules language. Proceedings of 13th International World Wide Web Conference (WWW), 2004.
13. Horrocks, P. Patel-Schneider, F. vanHarmelen: ¿From SHIQ and RDF to OWL: The Making of a Web Ontology Language. WebSemantics, 1, Elsevier, 2004.
14. F. Kauferand M. Klusch: Hybrid Semantic Web Service Matching with WSMO-MX. Proc. 4th IEEE European Conference on Web Services (ECOWS), Zurich, Switzerland, IEEE CS Press, 2006
15. U. Keller, R. Lara, H. Lausen, A. Polleres, D. Fensel: Automatic Location of Services. Proceedings of the 2nd European Semantic WebConference (ESWC), Heraklion, Crete, LNCS 3532, Springer, 2005.
16. M. Klusch, B. Fries, K. Sycara: Automated Semantic Web Service Discovery with OWLS-MX. Proc. 5th Intl. Conference on Autonomous Agents and MultiAgent Systems (AAMAS), Hakodate, Japan, ACM Press, 2006
17. M. Klusch, Z. Xing: SemanticWeb Service in the Web: A Preliminary Reality Check. Proc. First Intl. Joint ISWC Workshop SMR2 2007 on Service Matchmaking and Resource Retrieval in the SemanticWeb, Busan, Korea, 2007.
18. S. Narayanan, S. McIllraith: Simulation, verification and automated composition of WebServices. Proc. of 11th International Conference on the WorldWide Web (WWW), Hawaii, 2002.

19. S. McIllraith, T.C. Son: Adapting Golog for composition of Semantic Web Services. Proc. International Conference on Knowledge Representation an dReasoning KRR, Toulouse, France, 2002.
20. J. Pan, I. Horrocks: RDFS(FA): Connecting RDF(S) and OWL DL. IEEE Transactions on Knowledge and DataEngineering, 19(2):192-206, 2007.
21. Preist: Semantic Web Services— Goal sand Vision. Chapter 6 in [24], 2007.
22. T. C. Przymusinski: On the declarative and procedural semantics of logic programs. Automated Reasoning, 5(2):167-205, 1989.
23. M. Stollberg, U. Keller, H. Lausen, S. Heymans: Two-phase Web Service discovery based on rich functional descriptions. Proceedings of European Semantic Web Conference, Buda, Montenegro, LNCS, Springer, 2007.
24. R. Studer, S. Grimm, A. Abecker (eds.): Semantic Web Services. Concepts, Technologies, and Applications. Springer, 2007.
25. S.Tobies: The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics. Artificial Intelligence Research (JAIR), 12, 2000.
26. K. Ross, J. S. Schlipf: Thewell-founded semantics for general logic programs. ACM, 38(3):620-650, 1991.
27. G. Yangand M. Kifer: Well-Founded Optimism: Inheritance in Frame-Based KnowledgeBases. Proceedings of 1st International Conference on Ontologies, Data bases and Applications of Semantics (ODBASE), Irvine, California, 2002.
28. Д.А. Івченко: Використання Neo4j і GraphQL на базі GraphDB для створення мікросервісу/ Д. А. Івченко. // System Analysis and Information Technologies – 2018. – С. 171–174.